Fast combinatorial search algorithm for the TrackML Particle Tracking Challenge

A1. Authors

Sergey Gorbunov[1,2]
[1)] Johann Wolfgang Goethe-University Frankfurt am Main
[2)] FIAS Frankfurt Institute for Advanced Studies
mail: *sergey.gorbunov.32@gmail.com*

A2. Team in the competition

- Competition Name: TrackML
- Team Name: Sergey Gorbunov
- Private Leaderboard Score: 0.89353
- Private Leaderboard Place: 3

A3. Summary

The algorithm is a variation of traditional combinatorial track finding algorithms, which use a local track model.

Unlike global algorithms such as Hough Transform method, these algorithms let particle trajectory to deviate from an ideal global helix, being flexible enough to follow all local detector features like non-uniform magnetic field, scattering in a material etc.

A price for the flexibility is the execution time and complexity. One needs a precise tool to follow a local trajectory and to distinguish between good and bad track candidates. This tool is typically the Kalman Filter. It is heavy, slow and complicated. It involves matrix operations, numerical methods for solving differential equation for particle movement; it needs detailed description of magnetic field and materials.

Therefore it is interesting to see what one still can do in a very complicated silicon detector without having this tool available.

As an accurate fit of trajectories can not be constructed, the only way to have a local trajectory model is to create local helices using few local measurements and ignore the rest of the measurements. This approach appears to work pretty well. The best results have been achieved by using only 3 neighbouring measurements for the trajectory creation. This 3-hit trajectory model seems to be flexible enough to be able to follow all local features. At the same time it is accurate enough to not to loose the global trajectory due to chaotic local deviations.

Although the algorithm is designed to be fast, and even is called “fast”, it is not yet fast. It has been tuned for the first „accuracy phase“ of the TrackML competition and still needs to be optimised in speed.

The only external tool has been used is ROOT. It has been used for data analysis, not for the reconstruction. One needs to install it in order to compile the code, but potentially this dependency can be removed. ROOT should be installed on the system; a path to “root-config” macro should be in the PATH. Version 6.14 has been used, but the code will compile with any other version.

The test data has been reconstructed on a Mac laptop with 2,6 GHz Intel Core i5 processor and 8GB RAM. The reconstruction takes approximately 1.2 minutes per event.

A4. High-level Description

The algorithm consists of three parts. The first part is „tracklet construction“. It deals with hit-to-hit combinatorics and creates short tracklets on pre-selected detector layers. The second part is „tracklet prolongation“. It creates „track candidates“ by extending tracklets to other layers and collecting their hits there. The third part is sorting of the found track candidates.

For each detector layer a nominal 2-dimentional detector plane is created. It has 2 coordinates: ϕ and T. Here ϕ is the polar angle in the range (+-π). T is the second dimension which corresponds to Z for radial layers and R for forward/backward layers. All the hits are projected on their detector planes. There are two options here: one can use direction to the origin for the projection or just a hit coordinate (z or r correspondingly).

Hits are sorted on the layers according to they (ϕ,t) coordinates. For fast data access there is a regular 2-D grid created on every detector layer where hits are stored in corresponding grid bins.

The algorithm flow:

1. Tracklet construction

   It is performed on 3 (optionally, 2) selected layers

   1a. Every hit1 is taken from layer1 (optionally, hit1 can be the origin (0,0,0))
   1b. A straight line which connects the origin and the hit1 is projected to a layer2. Within some (ϕ,t) - search window every hit2 is taken.
   1c. A straight line, which connects hit1 and hit2 is projected to the next layer – layer3. Again, within some (ϕ,t) - search window every hit3 is taken.
   1d. A helix is constructed on hit1, hit2, hit3. On the XY plane the helix crosses all the three hits, in z it goes through hit2 and hit3. Then a z-deviation of the helix from the hit1 is investigated. If it is too large, the hit combination is rejected, otherwise the three hits are stored as a tracklet and the prolongation step starts.

2. Tracklet prolongation

   Tracklet is prolonged to the next detector layer along its trajectory. The closest hit is found.

   2a. When the found hit is close enough, it is added to the tracklet. The tracklet trajectory is recreated using the new hit and two hits from two previous layers. With the new trajectory one performs a search for duplicated hits on the layer.

   2b. When there is no good hit found and the prolonged trajectory is in the inner part of the layer (which is the layer size minus some margin), a hit is considered to be „missing“. When more than one hits are missing, the prolongation stops.

   2c. In case a good closest hit is found and all the duplicates are picked up, but still there are some hits left in the search area, one can create another search branch. To do so a copy of the tracklet is created with different hit on this layer. This duplicated

tracklet is then processed separately as any other tracklet. The branching is realised in an efficient way with almost no computing overhead. But practically, the creation of branches does not improve efficiency at all, and currently is switched off.

After the layer is processed, the tracklet is prolonged to the next layer, and so on. The prolongation is also performed in inward direction, towards the origin. When hits on all the layers are collected, the tracklet is stored in a list of track candidates. When all tracklets are processed, the algorithm goes to the track selection step, the step 3.

3. Selection of good tracks.

   It is done in a simple way. In the list of track candidates the best candidate is identified. It should have more hits then the others or the same amount of hits, but smaller average deviation of its hits from its trajectory. The best candidate is stored as a „track"; its hits are removed from the plane. Then the search for the next best candidate is performed, and so on. The search stops when the next best candidate has fewer hits than it is required (it is a parameter).

   This part of the algorithm is the slowest one. It has quadratic time dependency and needs optimisation in speed, which can be done without much effort.

4. The algorithm starts again from step 1 with different set of 3 (or 2) base layers.

A5. Scientific details

1. A physical trajectory model and the magnetic field

   A physical trajectory model is used (x,y,z,px,py,pz,q). Here (xyz) is a spatial position of a trajectory point, (px,py,pz) are three components of a particle momentum, q =+-1 is a charge.

   In the first version, the magnetic field was set to 1 in calculations, such, that the particle momentum in XY was equal to its radius in cm. Basically, the model was not different from any geometrical helix parameterisation. It is just more convenient to have a parameterisation point directly on the trajectory instead of parameterising some center of a circle 10 kilometres away.

   Then it appears, that the field is changing significantly from layer to layer, and one should take this feature into account during prolongation. For that propose the filed was fitted on each layer using the „truth" data and some simple field approximation model, which can be found in the code.

   In the newest version, for every hit its approximate field value is calculated and stored directly in the hit data structure. There are 3 different approximate field values: two for forward & backward prolongation and one more for a helix construction.

2. Organising the data: sorting of hits on detector layers.

   There is a regular 2-D grid created on each layer for extremely fast hit search within a predefined search window. The details can be found in the code in „FitLayer" class. To create the grid one only needs to loop twice over the hits. And it does not involve any pointer operations or memory allocation calls.

To search for hits one just calculates two bin numbers by doing a couple of multiplications, and then looks into 4 neighbouring bins only.

An interesting feature: the grid overlaps in ϕ. Hits which stay close to ϕ=+-π value are duplicated and are present on both +-ϕ edges of the grid in order to cover this region without introducing extra if() branches in the code.

A6. Interesting findings

1. Branching does not help at all. Despite the fact that there are over 10000 particles in an event, their trajectories seem to be well separated in space. Another explanation is that there are too many track candidates constructed and they simply cover all possible local branches. Another explanation could be, of course, that there is a bug in the code.
2. A local track model with last-3-hits-helix is good enough for constructing track candidates. May be for the final track selection one should still apply something more sophisticated.
3. Local variations of the magnetic field should be taken into account, one way or another. One can consider them as scaling factors for the curvature in a track model.
4. A manual selection of cuts and search windows is definitely not a right way to go. It should be replaced with some automatic procedure.
5. What is the most surprising: the usual helix track model works well not only for radial layers, but also for forward and backward regions, where the track projections on XY plane can be very short and very distorted.

A7. Simple Features and Methods

One can achieve 90% of performance for 10% of the execution time by creating tracklets only in the vertex region, with a constraint to the origin.

A8. Model Training and Execution Time

An analogue for the model training here is a geometry analyser. It calculates layer sizes and magnetic field values, or in other words, trajectory prolongation coefficients, for each layer. It takes less than a second per event.

The execution time is approximately 1.2 minutes per event on a single 2,6 GHz CPU. The algorithm is not optimised in memory usage. Currently it fits to 300MB.

A9. Outlook

A weak point of the algorithm is an absence of good criteria for rejection of fake track candidates. The tracklet constructor produces many good and fake track candidates. Most of them should stay in the heap of the candidates till the end of the reconstruction. Due to the poor fake rejection criteria, one cannot remove hits from the already found tracks early. Therefore each track is in fact found many times, each time starting from a new tracklet on new base layers. One has to investigate a structure of the fake tracks and try to identify more good tracks after each tracklet construction step.

A10. References

Discussion forum: https://www.kaggle.com/c/trackml-particle-identification/discussion