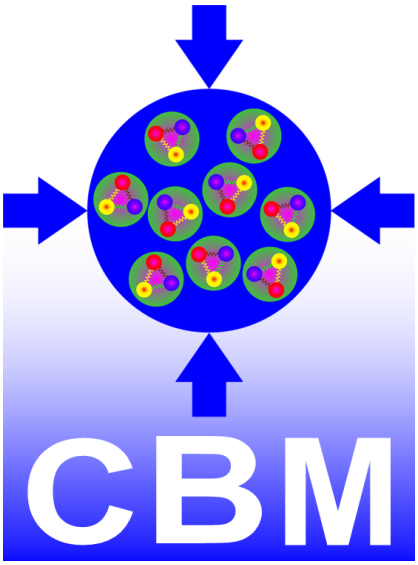
Eötvös Loránd Tudományegyetem

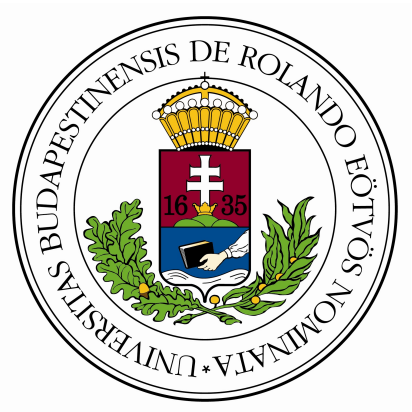
Kari Tudományos Diákköri dolgozat

A CBM/FAIR szimuláció használata és részecske klaszterezés nehézion ütközésekben

Olar Alex - Fizika BSc III.

Témavezető: Wolf György





Kivonat

Egy hónapot töltöttem a nyár folyamán, július hónapba, Darmstadtban a GSI nevű kutatóközpontban. Kint tartózkodásom célja az volt, hogy többet megtudjak a CBM saját szimulációjáról, amely kutató csoport már az épülő FAIR[1] része. Ezalatt a hónap alatt megismerkedtem mélyebben a ROOT [2] nevű szoftverrel, a helyi cbmROOT-tal [3], valamint a C és C++ programozási nyelvekkel.

A kint létem alatt sokat tanultam a detektor technológiákról, valamint az azokban lejátszódó eseményekről.

Itthoni munkám során a nehézion ütközések szimulációjához kapcsolódva egy klaszterező program fejlesztésével foglalkoztam, ami a kinyert adatokat csoportosítja térbeli és impulzustérbeli távolságuk alapján, előre definiált klaszterezési mérettel, az MST algoritmus felhasználása segítségével.

Tartalomjegyzék

[1] Facility for Antiproton and Ion Research
[2] CERN szoftver részecske analízishez
[3] CBM (Compressed Barionic Matter) szoftver a CBM/FAIR által fejlesztve

I.. Alapok

I.1. QCD - BSc-s szemmel

A 20. század folyamán fizikusok szembesültek azzal, hogy milyen abszolút fontos szerepet töltenek be a szimmetriák az univerzum és a körülöttünk lévő világ megismerésében, miután a megmaradási tételekből következtettek rájuk.

A kvarkok felfedezése végre rendet teremtett a részecske állatkertben (particle ZOO), ahogy az elemi részecskék folyamatosan növő számára Niels Bohr szellemesen referált. A mennyiséget ami a különböző kvarkokat bizonyos szempontból jellemzi *íz*nek hívjuk, kezdetben csak három kvarkíz volt ismert, úgy mint: u (up), d (down), s (strange). A hadronok két csoportba oszthatók szét: *mezonok* és *barionok*, amelyek rendre egy kvark-antikvark párt vagy három kvarkot tartalmaznak.

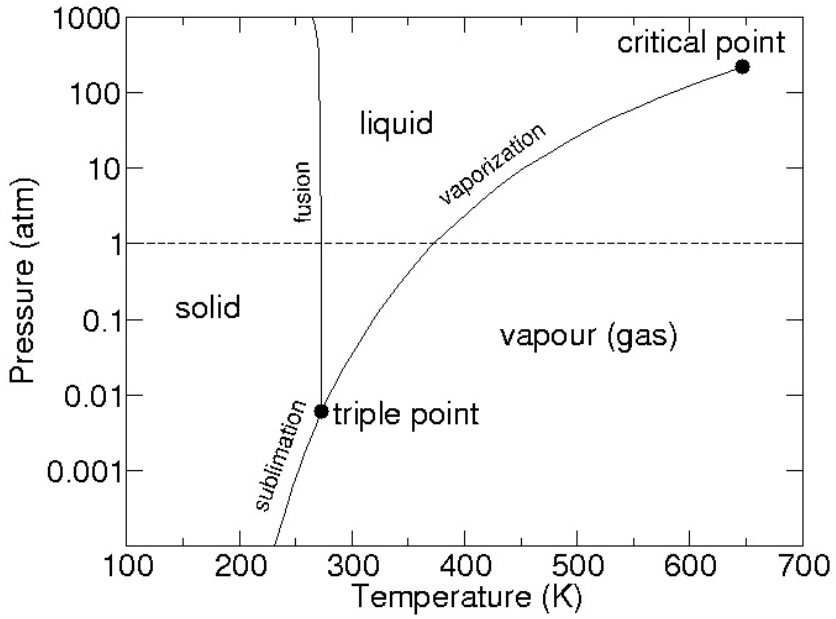
Az erős kölcsönhatás, ami a kvarkok között ható elemi kölcsönhatás, egyedi tulajdonsága a bezárás, ami megakadályozza a kvarkokat abban, hogy elszeparálva, izoláltan megtalálhatóak legyenek. Az erős kölcsönhatás töltését színnek hívjuk. A bezárás miatt, az elemi részecskék csak úgynevezett semleges színben létezhet, amit gyakran 'fehérnek' nevezünk. Az erős kölcsönhatást leíró alapvető elmélet a Kvantumszíndinamika (Quantum Chromo Dynamics) - QCD.

A QCD elemi részecskéi a kvarkok és antikvarkok, amelyek a gluonok által hatnak kölcsön, melyek szintén színtöltést hordoznak. A gluonoknak 8 fajtájuk van, hogy minden színtranszformáció leírható legyen segítségükkel. A gluonok önmagukkal is kölcsön tudnak hatni.

I.2. CBM fizika

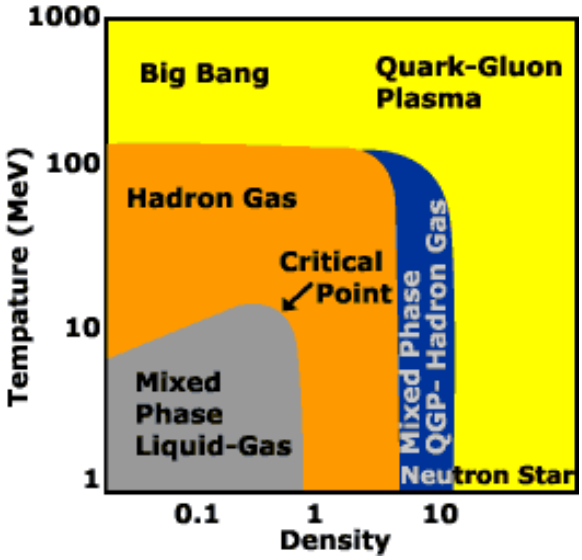
A barionanyaggal foglalkozva az elsődleges cél, hogy megértsük és jobban megismerjük a fázisdiagramot és magukat az átmeneti folyamatokat. Először is rövid bevezetőként egy kis termodinamikai áttekintéssel kezdek a fázisokról és a fázisátalakulásokról, alapul véve a The CBM Book-ot:

A víz fázisdiagramja megmutatja annak különböző fázisait egy nyomás-hőmérséklet diagramon. Ismeretes, hogy adott körülmények között van egy hármaspont, amelyben a víz mindhárom halmazállapotában előfordul. A fázisok közötti vonalak mentén a víz szintén több (itt kettő) halmazállapotban előfordulhat és ezek mindketten megtalálhatóak a megfelelő körülmények között. Elsőrendű fázisátmenetnek hívjuk, amikor ezen vonalakon 'áthaladva' halmazállapotváltozás történik. Továbbá megkülönböztetünk egy kritikus pontot is, amely után a fázisok nem különülnek el jelentős mértékben, ezután csak egy úgynevezett sima *crossover* figyelhető meg, nem elsőrendű fázisátmenet.

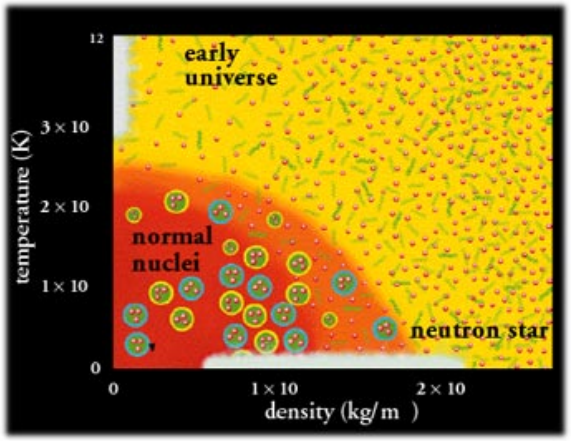


1. ábra. A víz fázisdiagramja.

Most, hogy gyorsan áttekintettem a víz fázisdiagramját, vagy legalábbis egy részét, ideje továbblépni és feltenni a kérdést, hogy mi a helyzet az erősen kölcsönható anyaggal. Az erősen kölcsönható anyag fázisdiagramja ugyanis elméleti stádiumban van, még nincs teljesen feltárva. Az ábrákon olyan különböző és elengedhetetlenül fontos fázisok vannak, amelyek a korai univerzumot jellemezték vagy éppen a neutron csillagok anyagát alkothatják. Itt mindkét ábrán egy hőmérséklet-sűrűség diagramot láthatunk.



(a) Hőmérséklet MeV-ban kifejezve, míg a sűrűség magsűrűségben van megadva, mindkét skála logaritmikus



(b) Ezen a diagramon a fázisok természetbeli előfordulását láthatjuk.

A fentebbi ábrákon is látható egy fázis, amit kvark-gluon plazmának nevezünk. Ez az állapot jelen volt a Nagy Bummnál, de később nem maradt fenn, a hőmérséklet hirtelen csökkenése miatt. Látható az is, hogy a neutron csillagok belseje is kvark-gluon plazmát tartalmazhat, de azokban nem a hőmérséklet kell igen magas legyen, hanem a csillag sűrűsége.

Nyilvánvalóan, a kvark-gluon plazma földi megfigyelésének egyetlen lehetőségét a nagy energiás részecskegyorsítók és az általuk létrehozott nehézionok ütköztetése biztosítják. A QCD jellemző tulajdonsága, hogy a kvarkok közti összetartás csökken, ahogy az ütközési energiát növeljük, ez az aszimptotikus szabadság jelensége. A részecskefizika egy másik fontos szimmetriája a kiralitással kapcsolatos. Ez lényegében azt jelenti, hogy egy tömegtelen részecske spinje és sebességének iránya egymáshoz képest milyen irányba mutat. Hogyha azok egyirányúak, akkor a részecske jobb kezes, ha ellentétes irányúak akkor pedig bal kezes. Tömeggel rendelkező részecskék esetén a kiralitásnak nincs értelme, mert az egy Lorentz-transzformációval ellenkezőjére változtatható. Mivel az up és down kvarkok tömege közel azonos és igen kicsi (nagy energiákon jó közelítéssel 0) ezért azt szokták mondani, hogy a QCD-nek körülbelüli királis szimmetriája van. Ellenben, ez spontán sérülhet alacsony hőmérsékleteken és sűrűségeken a párkölcsönhatások miatt. Emiatt az egyik királis irány ekkor gyakoribb lesz, mint a másik, ezt nevezzük lényegében királis szimmetriasértésnek.

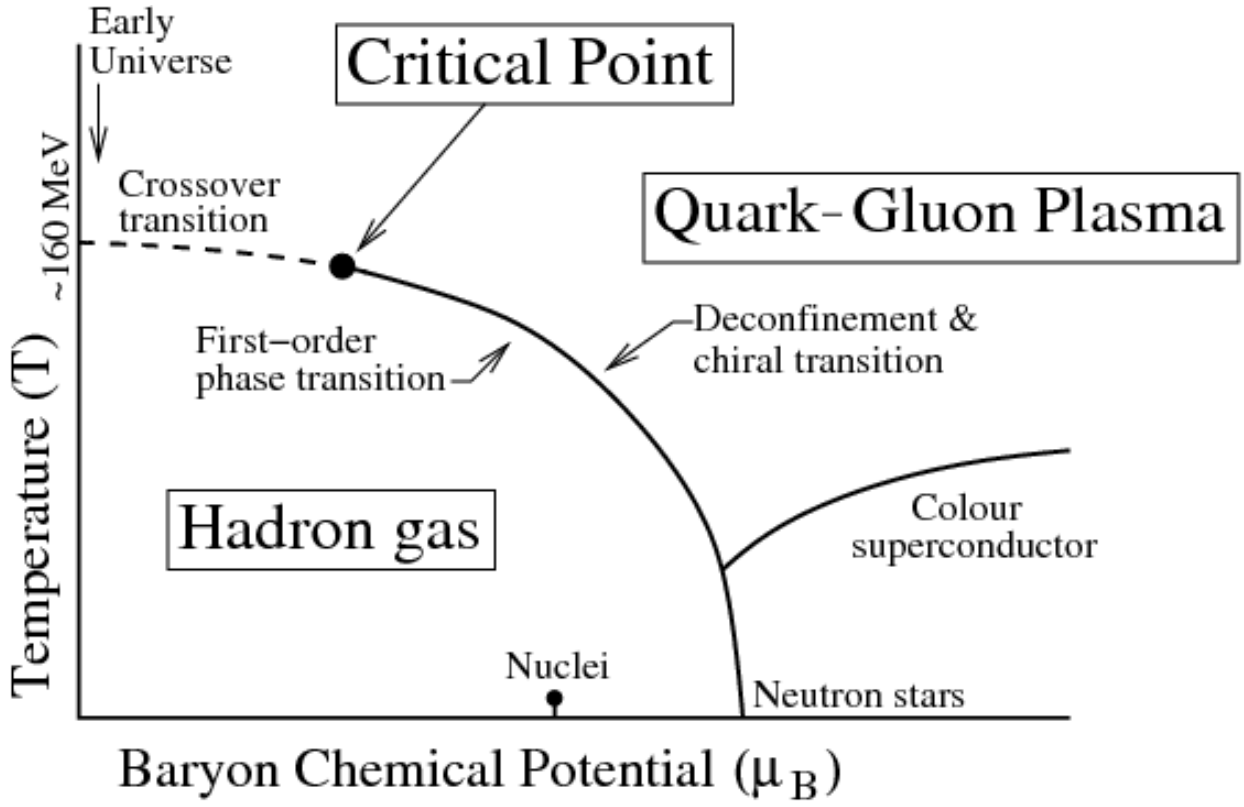
II.. A CBM detektor

II.1. Elmélet

A nehézionok ütközésének vizsgálata és az adatok feldolgozása egy borzasztóan komplex feladat a reakció tranziens természete miatt. Lényegében az a cél, hogy az ütközés során, 10^{-22} s-ig fennálló állapot segítségével következtessünk az erősen kölcsönható anyag fázisdiag-

ramjára, a jelenség természetére. Az idő természetesen nagyon rövid, és az egész jelenség csak a melléktermékek révén vizsgálható.

Az elmúlt évtizedben a fő tudományos tevékenység a témában a brookheaven-i RHIC [4] központban és a CERN-ben található LHC [5] gyorsítónál zajlott. Ezek a központok nagyon fontos, és érdemleges adatot biztosítanak a fázisdiagram vizsgálatához. Ellenben ezek mind az alacsony sűrűségű régiót vizsgálják és csak az aszimptotikus szabadságot tudják feltérképezni a hadron gáz és a kvark-gluon plazma között. A FAIR projekt ezekkel szemben sokkal magasabb barion sűrűséget tervez elérni, hogy lehetőséget biztosítson az első rendű fázisátmenet vizsgálatára, valamint a kritikus pont környékének feltérképezésére.



3. ábra. A fentebb említett elméleti fázisdiagram

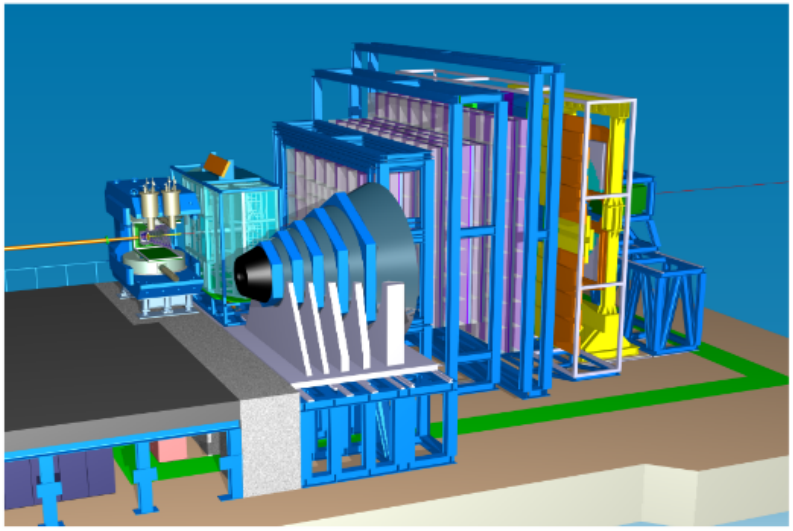
II.2. Detektor elrendezés és a szimuláció

A detektor elrendezése balról jobbra haladva a következő (ábra):

- CBM szupravezető mágnes szilícium spektrométerrel
- a micro vertex detektor (MVD) az előbbi belsejében
- a szilícium követő rendszer (STS) is
- Cserenkov-detektor (RICH - ring imaging Cherenkov detector - világos kék)

[4]Relativistic Heavy Ion Collider - Brookhaven
[5]CERN - Large Hadron Collider

- müon sprektrométer (fekete)
- ezt követi 4 réteg átmeneti sugárzás (TRD - transition radiation detector) detektor
- és egy time-of-flight (TOF) fal
- a fő detektorok után található még egy célfigyelő detektor (PSD)



4. ábra. A detektor elrendezés

A szilícium követő rendszer feladata az, hogy rekonstruálja majd a részecskék trajektóriáit. Csak töltött részecskék észlelésére képes, de képes mérni a töltés nagyságát és az impulzust is tud mérni. A TOF fal igen nagy felbontást tud elérni, nagyjából 60 ps-os felbontásra is lehetőség van.

A CBM projekt még egyenlőre csak terv szintjén létezik, a szimulációt folyamatosan fejlesztik. Jövőre, vagy legkésőbb 2019-re már tervben van egy miniCBM detektor építése az esetleg később felmerülő tervezési, kivitelezési problémák elkerülésére. A FAIR létesítmény építése idén nyáron kezdődött és az első részecskenyaláb 2022-ben várható. A miniCBM projekt a meglévő GSI gyorsítónál fog tevékenykedni az addig fennmaradó időben, ahol megpróbálják a számítógépfarmot tökéletesíteni, hogy az adatokat minél gyorsabban feldolgozhassák.

A FAIR tudósai kifejlesztettek egy több tízezer soros szimulációt, ami a ROOT-on alapszik. Ezt ők cbmROOT-nak hívják, mivel teljes egészében a CBM-hez igazodik és ingyenesen elérhető bárki számára. Sok jól ismert nehézion szimulációs eljárást használnak, amik a CBM környezetre vannak szabva, úgy mint: UrQMD [6], valamint PHSD [7]. Ezek a szimulációs kódok széles körben használtak nem csak itt, hanem az egész nehézion fizika területén.

[6]Ultra Relativistic Quantum Molecular Dynamics
[7]Parton Hadron String Dynamics

III.. A Φ-mezonról röviden

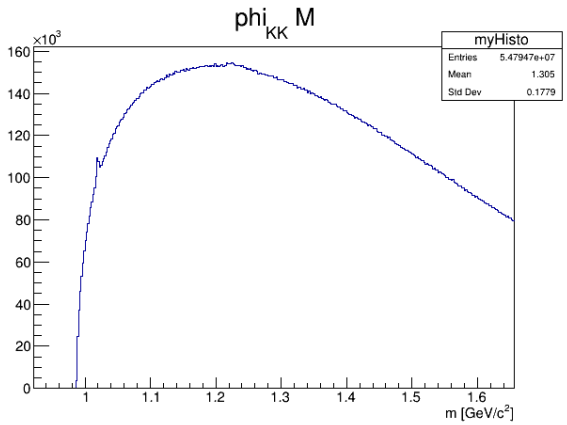
III.1. Φ-mezon rekonstrukció

A CBM detektor egy általános célú nehézion mérési eszköz lesz, hogy az erősen kölcsönható anyag fázisdiagramját vizsgálni lehessen. A rezonanciák nagyon fontosak, hogy a sűrű anyagot vizsgálni tudjuk az ütközés során. Az ilyen rezonanciák egyike ami fontos a CBM és a fázisdiagram vizsgálatának szempontjából pedig a Φ-mezon, aminek nagyon kicsi a hadronokra vett hatáskeresztmetszete így eléggé valószínűtlen, hogy kölcsönhat a nagy mennyiségű hadronnal, ami a reakció során keletkezik, vagyis jó indikátora a sűrű, kezdeti eseménynek. A Φ-mezon egy strange és egy anti-strange kvarkot tartalmaz és a kulcsa lehet az s kvark partonikus anyagban lévő keletkezésére. A Φ-mezon K^+, K^- párokra bomlik nagyjából 50%-os eséllyel és egyebekre (pl. dileptonokra is). A közepes élettartama nagyjából $1.55 \cdot 10^{-22}$ s tehát még a TOF falat sem éri el, csak a bomlástermékei lesznek detektálva már korábban is. A tömege 1.019 MeV ami a kaonok invariáns tömegével kifejezve egy rezonancia csúcsként látható az ütközés/szimuláció után kinyert adatok között. Ahol a kaonok invariáns tömege:

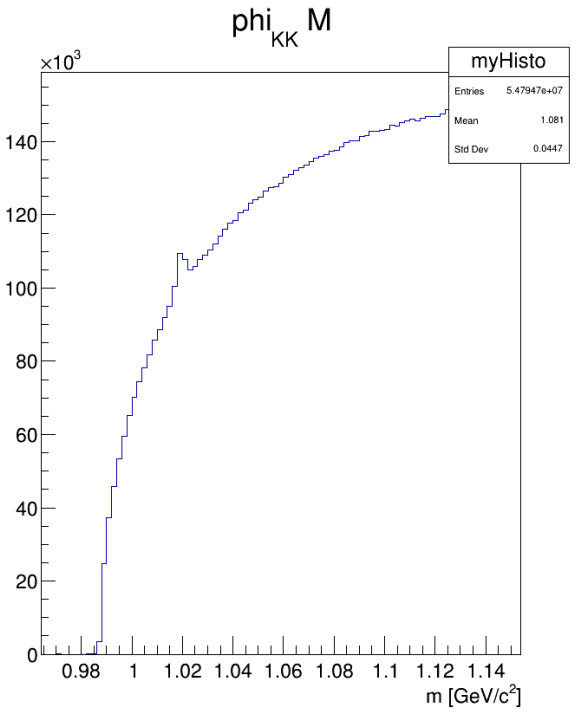
$$M_{KK} = \sqrt{(E_1 + E_2)^2 - (\underline{p}_1 + \underline{p}_2)^2}$$

Én a PHSD adatait vizsgáltam, amin lefuttattam a CBM szimulációt. Egy Au+Au centrális ütközést vizsgáltam 10 GeV-es bombázó energián. A CBM szimuláció kimenetét a cbmROOT-tal rekonstruáltam. Több mint 5 millió esemény szerepelt a kezdeti *.root* fájlban amit a szimulációhoz használtam.

A hisztogramokon az x-tengelyen a kaon párok invariáns tömege szerepel, az y-tengelyen pedig az adott energián a 'beütések' száma. Egy apró kiugrás látható a nagy kombinatorikus háttéren nagyjából 1.02 GeV környékén ami pontosan a Φ-mezonok bomlásából adódik.



(a) A kombinatorikus háttér és egy apró, de jól látható csúcs.

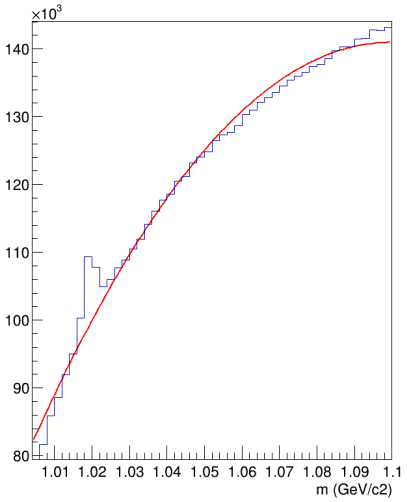


(b) A csúcs.

Egy másodfokú polinommal próbáltam becsülni a hátteret. Az illesztés paraméterei ($ax^2 + bx + c$) :

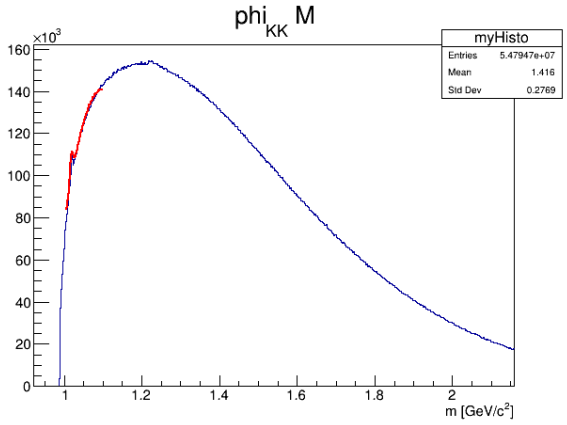
Parameter name	Value []	Error
a	-7.70559e+06	78750.5
b	1.42738e+07	150055
c	-6.49147e+06	71438.9

A háttér illesztése:

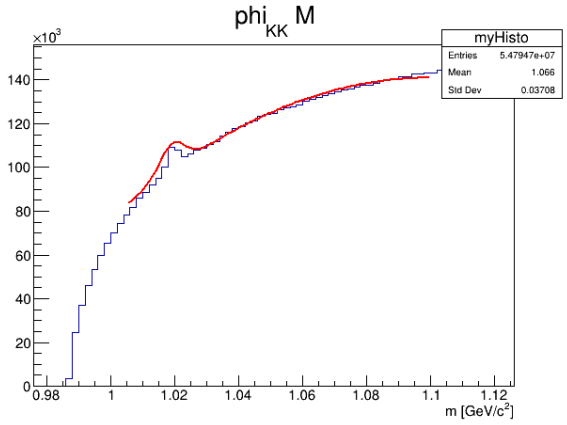


6. ábra. A háttérre vett illesztés a másodfokú polinommal.

A csúcs közelítéséhez más módszert alkalmaztam. Egy alacsony multiplicitású jelet használtam a csúcs alakjának becsléséhez, amit egy Gauss-függvénnyel illesztettem, majd ezt skáláztam fel a csúcshoz, az állandó nagyságú háttér mellett, az eredmények a következők:



(a) A háttér és a csúcs illesztése.



(b) Ráközelítve.

Itt a ROOT makróm, amit az 'illesztéshez' használtam:

```
void fit(){

    TFile* srcFile = TFile::Open("
    KFParticleFinder_phsdwocsr_auau_10gev_centr_sis100_electron_5M_ToF.root");
    TDirectory* phi = (TDirectory*)srcFile->Get("KFTopoReconstructor/
    KFParticlesFinder/Particles/phi_{KK}/Parameters");
    TDirectory* phi_signal = (TDirectory*)srcFile->Get("KFTopoReconstructor/
    KFParticlesFinder/Particles/phi_{KK}/Parameters/Signal");

    TH1F* M = (TH1F*)phi->Get("M");
    TH1F* Msignal = (TH1F*)phi_signal->Get("M");
    Msignal->SetName("Msignal");

    TFile* myFile = new TFile("phi_fit.root","recreate");
    TH1F* myHisto = (TH1F*)M->Clone();
    myHisto->SetName("myHisto");
    TH1F* myBackground = (TH1F*)M->Clone();
    myBackground->SetName("myBackground");
    TH1F* mySignal = (TH1F*)Msignal->Clone();
    mySignal->SetName("mySignal");
    myFile->cd();

    srcFile->Close();

    TCanvas* canv = new TCanvas("canv","Total fit",640,480);

    TF1* background = new TF1("background","pol2",1.004,1.1);
    TF1* signal = new TF1("signal","gaus",1.011,1.033);

    TF1* total = new TF1("total","gaus(0) + pol2(3)",1.005,1.1);

    background->SetParameters(0.2,1.01,108000.);
    background->SetParNames("landau_1","landau_2","landau_3");
```

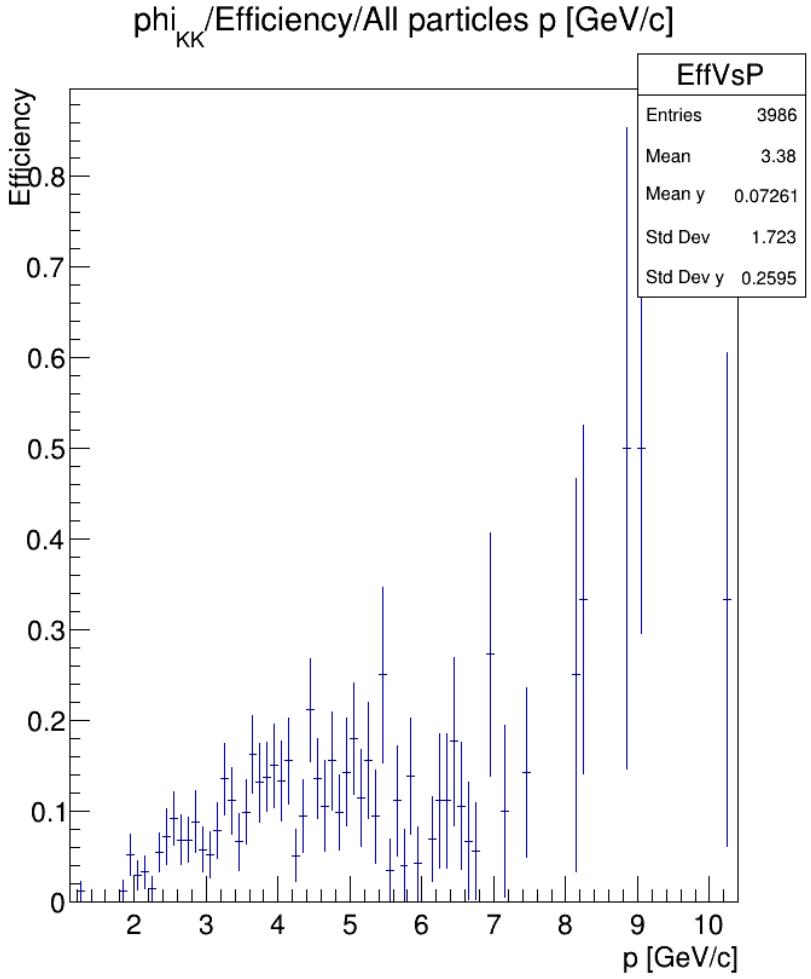
```
31      myBackground->Fit("background","R+");
32
33      Double_t params[6];
34      background->GetParameters(&params[3]);
35
36      signal->SetParameters(3200., 1.021, 3000.);
37      signal->SetParNames("scale","mean","sigma");
38      mySignal->Fit("signal","R+");
39      signal->GetParameters(&params[0]);
40
41      params[0]*=250.; // rescale
42      total->SetParameters(params);
43      total->SetParNames("scale","mean","sigma",
44            "la","b","c");
45      myHisto->Draw();
46      total->Draw("same");
47
48      signal->Write();
49      background->Write();
50      total->Write();
51      myHisto->Write();
52      canv->Write();
53
54      myFile->Close();
55
56  }
```

Ez nem igazán pontos közelítés, de jól szemlélteti, hogy a háttér jól közelíthető ellenben ekkora számú eltérés már megmutatkozik benne.

III.2. Φ-mezon a CBM-ben

Igen nehéz feladat lesz hatékonyan detektálni a Φ-mezonokat a CBM detektorrendszerrel. A részecskék nem csak rövid életűek, de egy hatalmas háttér is nehezíti az apró csúcs megtalálását. Ezért is kell hatalmas számú eseményt vizsgálni, hogy a csúcs a statisztikában már látható legyen. Ennek ellenére határozottan mondhatjuk, hogy a CBM detektor képes lesz a Φ-mezonok detektálására és ezáltal a strange kvark termelődésének megértésére az erősen kölcsönható anyagban.

A szimuláció hatékonysági mutatókat is biztosít. Mindezeket különböző részecske impulzusok esetén. A jelzett detektálás hatékonysági értékek nem túl magasak, de eléggé stabilak adott tartományokban az észleléshez:



8. ábra. Hatékonyság az impulzus függvényében

IV.. A szimuláció

IV.1. Telepítés

A CBM szimuláció telepítésének három fő komponense van, az egyik a FairROOT, majd a FairSoft és végül a cbmROOT. Bármilyen rendszerre telepíthetőek az alábbi linkről: `https://redmine.cbm.gsi.de/projects/cbmroot/wiki/InstallCbmRootAuto`

Erősen ajánlott a telepítést ezt követve megtenni, mivel rengeteg apró, de akadályozó probléma előjöhet a telepítés során. A teljes csomag tartalmazza a ROOT-ot is, így az egész nagyjából 25 GB helyet foglal.

IV.2. Bevezetés

Maga az ütközés a UrQMD és a PHSD programok segítségével játszódik le, a CBM szimuláció a detektor választ szimulálja, tehát az ezekből származó adatokat kapja meg kezdeti paraméternek. Ezek a modellek az ALICE, RHIC és LHC detektornak, valamint nem utolsó sorban a CBM detektornak lettek fejlesztve. Én főleg UrQMD adatokat használtam, de PHSD fájlokkal is találkoztam kint létem során.

Az első lépés az, hogy le kell futtatni egy Monte Carlo szimulációt, hogy képeset legyünk a 'valódi' adatokat összepárosítani a keltett eseményekkel. A program ezen része arra lett tervezve, hogy kiszűrje a találatokat a detektor anyagban és olyan pontokat találjon, amelyek később trajektóriákká összeállíthatók.

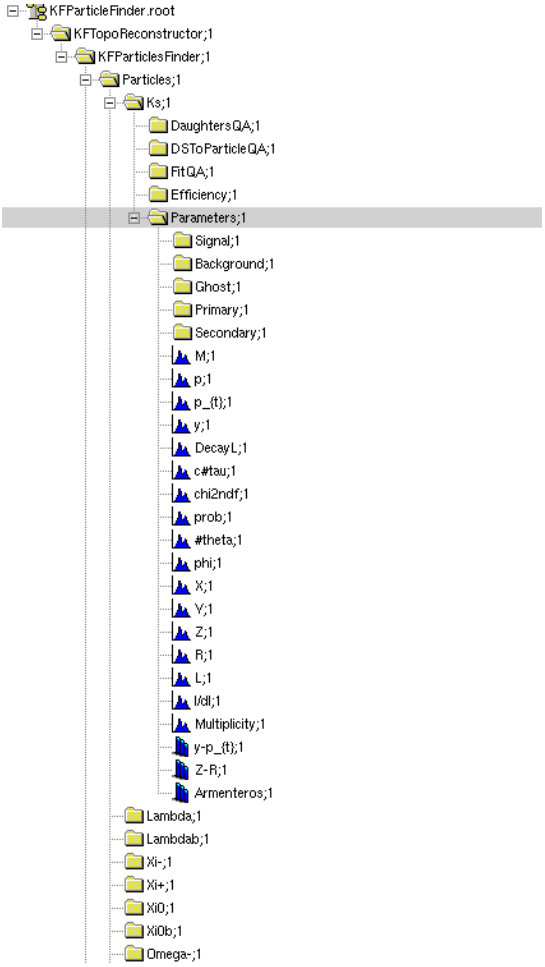
A program a Geant3 és Geant4 programokat használja, hogy a részecskék anyagon való áthaladását szimulálja. Ez is a Monte Carlo szimuláció része.

Az első makró kimenetén tehát egy szimulációs fájl van, ami az STS és az MVD detektorok által detektált találatokat tartalmazza valamint a TOF fal és egyéb detektorok adatait is. Ezeket felhasználva lép a program a második fázisba, a rekonstrukció részhez. A rekonstrukciós kód először is klasztereket próbál találni az MVD detektorban, hogy megtalálja, hogy hol volt az ütközés/ütközések kiinduló pontja. Ha ezt megtalálta továbbhalad és megpróbálja lekövetni a részecske pályákat. A töltött részecskék körpályára állnak az erős mágneses tér hatására így a pontokra köríveket próbálnak illeszteni és a legjobb illesztéssel bírókat fogadják csak el (van egy százalékos határ, ami alatt hibás detektálásnak ítélik). Én főleg az MVD és STD detektorokra koncentráltam, tehát a többit most nem említem itt.

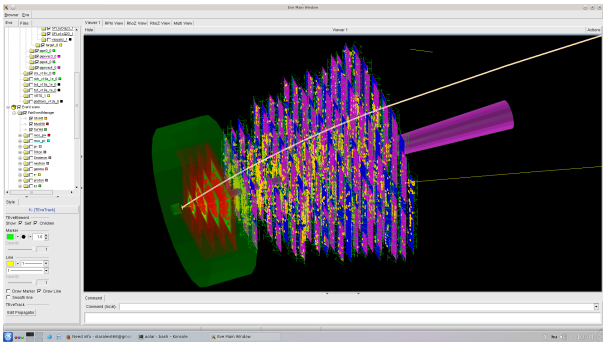
Nyilvánvalóan, a találatok és a pályákat többször próbálja meg a program helyesen megtalálni, azért, hogy elkerülje a hibákat. Kisebb az esélye így a hibás találatnak, vagy a hibásan illesztett trajektóriának. Ennék része a digitalizáció, ami lényegében azt jelenti, hogy a szimulációs program megpróbálja a detektor választ is számításba venni. Vegyük például az STS detektort. Ennek egy szálas, hálós elrendezése van, amikor egy részecske áthalad, akkor több szálban is detektáljuk, ezek metszéspontjában van a tényleges helye. De ha egyszerre két részecske ment át 'ugyan azon a ponton', akkor ezt nem láthatjuk, később a pályák illesztésénél probléma lehet. Ezért is van az, hogy ha az STS detektor több, mint 5%-a detektál, akkor a rendszer lényegében nem mér, nem szerez kiértékelhető adatokat.

A sikeres rekonstrukció után, ami a nyers adatokból létrehozta végső soron a trajektóriákat az egyetlen visszamaradó feladat a részecske felismeres és ezek pályákhoz való párosítása. Erre egy robusztus és hatékony program áll rendelkezésre, aminek a neve KFParticleFinder.

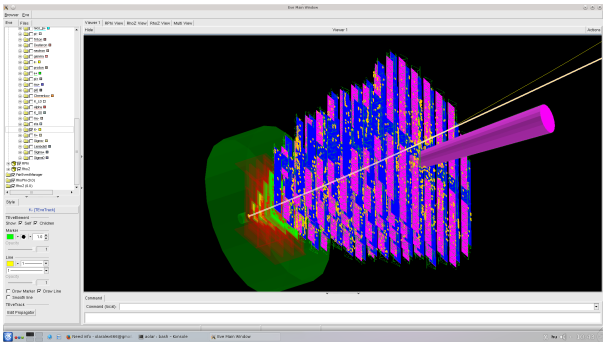
Ennek a programnak a kimenete egy .root fájl, ami rengeteg részecskét és hozzájuk tartozó adatot tartalmaz, detektálási hatékonyságról, háttérről, armenteros diagramokkal, bemenő és kimenő jelekkel, stb. . Az szerkezete nagyjából így néz ki:



9. ábra. A ROOT fájl struktúrájának egy része.



(a) A rekonstruált pályák az MVD és STS detektorokban.



(b) Másik szögből

IV.3. How-tos

Ahogy korábban említettem először a Monte Carlo szimulációt kell használni valamilyen bemeneti fájllal. Ez egy .root fájl vagy egy egyszerű ASCII fájl is lehet, a szimulációs kód

képes mindkettő fogadására. Egy ilyen fájlban részecske ID-k és impulzusuk található. A kimenete a PHSD és a UrQMD szimulációknak általában egy .root fájl, de például a HIJING sima szöveges kimenetet produkál. A CBM szimulációnál különböző függvények teszik lehetővé mindkét adattípus feldolgozását.

Megtanultam használni a jelgenerátor programot, amivel bárki, bármit küldhet a detektor szimuláció bemenetére. Én főként arra használtam, hogy kontrollált körülmények között, csak Φ-mezonokat küldjek be, amivel vizsgálni lehet, hogy mi lesz a program kimenetén a KFParticleFinder által kiadott .root fájlban. Ahhoz, hogy a generátor által biztosított ASCII fájlt olvasni tudja a szimuláció a következő módosítások szükségesek:

```
1   1  0  0  0
333  0.349404  0.108345  2.17087
1   2  0  0  0
333  -0.601515  -1.42376  7.32593
1   3  0  0  0
333  0.604993  0.756893  8.0675
1   4  0  0  0
333  -0.605273  0.957298  2.78006
1   5  0  0  0
333  -0.561403  -0.245707  1.30767
1   6  0  0  0
333  -0.111909  0.297546  0.780414
1   7  0  0  0
333  0.479322  0.647613  1.23907
1   8  0  0  0
333  -0.495742  -0.65654  1.05797
1   9  0  0  0
333  -0.736586  -0.211334  2.19586
1   10  0  0  0
333  0.0558235  -0.109982  3.04292
```

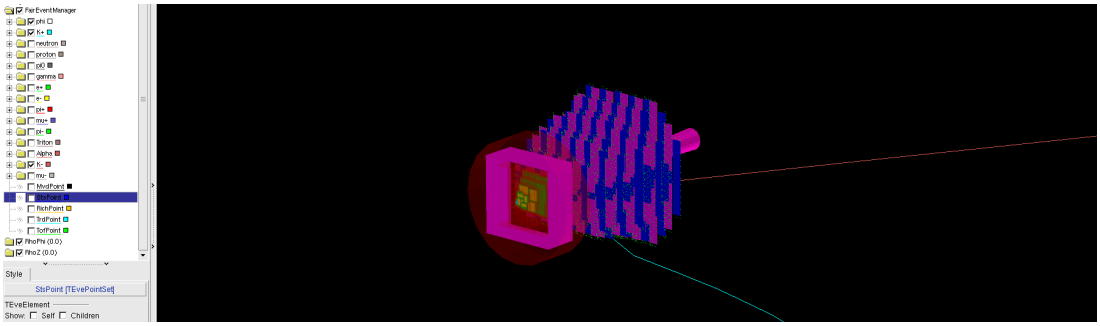
333 a részecske ID a Φ-mezonnál. Ezután a szimuláció tudni fogja, hogy hogyan dolgozza azt fel, és képes lesz azt elbomlasztani a megfelelő valószínűségekkel.

```
FairAsciiGenerator *SignalGen = new FairAsciiGenerator(inFile);
primGen->AddGenerator(SignalGen);
```

Fentebb a .root fájlokhoz használt *CbmUnigenGenerator* helyett ASCII fájlok esetén ezt kell használni. Még egy fontos lépes van itt. Ha szeretnénk vizualizálni a későbbiekben az eredményeinket, akkor engedélyeznünk kell a trajektóriák ilyen szintű mentését. Ez nyilvánvalóan nem hatékony hatalmas részecske számok esetén, de ha csak néhány részecskét küldünk be, akkor hasznos lehet látni, hogy hogyan is működik a program, esetleg hibákat is észrevehetünk.

```
// -Trajectories Visualization (TGeoManager Only )
run->SetStoreTraj(kTRUE);   //->
// --------------------------------------------------
```

Tehát a rekonstrukció után, valamint a részecske felismerés végeztével, ha bekapcsoltuk a vizualizációt képesek vagyunk vizualizálni az eseményeket. Ehhez az *eventDisplay.C* makrót kell futtatnunk. Ez a makró az egész CBM geometriát tartalmazza, tehát az egész detektort átláthatjuk vele. Megjeleníthető benne az összes trajektória és a részecskék. Néhány kép arról, ahogy egy Φ-mezon két kaonra bomlott:



11. ábra. Vizualizáció az MVD és STS detektorokban

IV.4. Φ-mezonok generálása és a kimenő adatok elemzése

A jelgenerátorral 2500 eseményt generáltam ahol a Φ-mezonok pont a céltárgy közepében helyezkedtek el, tehát mintha éppen ott keletkeztek volna az ütközés során. Az ilyen adatok elemzése azért fontos, mert ekkor kontrollált körülmények között, adott részecske számmal tudjuk vizsgálni a kimenő részeskék számát, eloszlását és ebből ismeretlen mennyiségű kezdeti részecskénél következtethetünk a Φ-mezonok számára így a strange keltés folyamatára.

A generátor makróban a bemenő nyaláb energiáját is változtathatjuk, valamint a környezet hőmérsékletét is (mindkettő GeV-ben) és persze azt is, hogy milyen részecskét akarunk generálni.

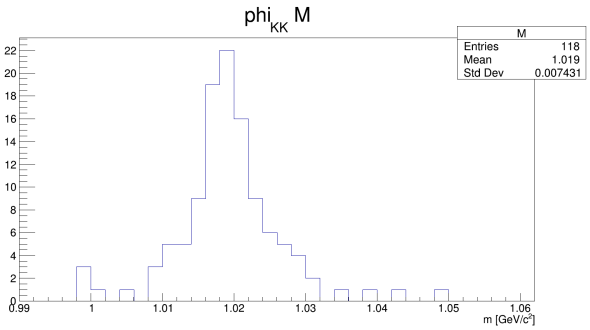
```
double fSlope = .154; // temperature
...
double eBeam = 10.; // beam energy
double pBeam = TMath::Sqrt(eBeam*eBeam - kProtonMass*kProtonMass);
...
 const int NParticlesPerEvent = 1;
 const double kSignalMass[NParticlesPerEvent] = {1.019455};     //  mass in GeV
 const int    kSignalID[NParticlesPerEvent] =  {333};
 ...
   for (int i=0; i<NEvent; i++){
   // Generate rapidity, pt and azimuth
   outputfile<<NParticlesPerEvent<<"   "<<i + 1<<"   "<<0.<<"   "<<0.<<"   "<<0.<<
    endl;
   for(int j=0;j<NParticlesPerEvent;++j) {
   double yD   = gRandom->Gaus(fYcm, fRapSigma);
   double ptD  = fThermal[j].GetRandom();
   double phiD = gRandom->Uniform(0., kTwoPi);

   // Calculate momentum, energy, beta and gamma
   double pxD    = ptD * TMath::Cos(phiD);
   double pyD    = ptD * TMath::Sin(phiD);
   double mtD    = TMath::Sqrt(kSignalMass[j]*kSignalMass[j] + ptD*ptD);
   double pzD    = mtD * TMath::SinH(yD);

   outputfile<<kSignalID[j]<<"   "<<pxD<<"   "<<pyD<<"   "<<pzD<<endl;

   }
   }
```

Jól látható, hogy ezt a makrót elég könnyű személyre szabni, tehát bárki könnyedén elkészítheti magának a számára megfelelő bemeneti fájlt. A bemeneti fájlt az események számával és a fájl nevével át kell adni a szimulációs programnak.

```
void run_mc_phi(TString inFile="Signal_phi_2500.txt", const char* setupName = "
    sis100_electron", Int_t nEvents = 2500)
{
  TString outFile = "sim_phi_2500.root";
  TString parFile = "param_phi_2500.root";
...
  // --- Define the target geometry ---------------------------------------
  //
  // The target is not part of the setup, since one and the same setup can
  // and will be used with different targets.
  // The target is constructed as a tube in z direction with the specified
  // diameter (in x and y) and thickness (in z). It will be placed at the
  // specified position as daughter volume of the volume present there. It is
  // in the responsibility of the user that no overlaps or extrusions are
  // created by the placement of the target.
  //
  TString  targetElement   = "Gold";
  Double_t targetThickness = 0.025;  // full thickness in cm
  Double_t targetDiameter  = 2.5;    // diameter in cm
  Double_t targetPosX      = 0.;     // target x position in global c.s. [cm]
  Double_t targetPosY      = 0.;     // target y position in global c.s. [cm]
  Double_t targetPosZ      = 0.;     // target z position in global c.s. [cm]
  Double_t targetRotY      = 0.;     // target rotation angle around the y axis
    [deg]
}
```

A kimenet egy .root fájl, ami mint már korábban említettem adatokat tartalmaz a beütésekkel a detektor anyagban. Fontos megemlíteni, hogy a céltárgyat is bárminek definiálhatjuk, a helyzetét is változtathatjuk, de a mi feladatunk, hogy helyesen tegyük, mert a szimuláció lefut úgy is, hogy a nyaláb el sem találja a céltárgyat.

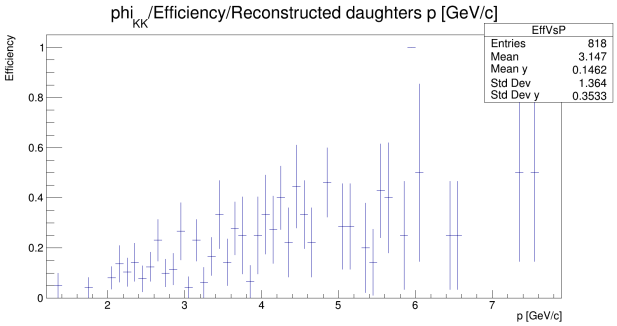
Ezután a rekonstrukciós fájlt is kissé módosítani kell, majd ez a trajektóriákat találja meg. Majd a fizika makrót kell futtatni, hogy a KFParticleFinder megtalálja a pályákhoz tartozó részecskéket. A kimeneti .root fájlból néhány részlet:



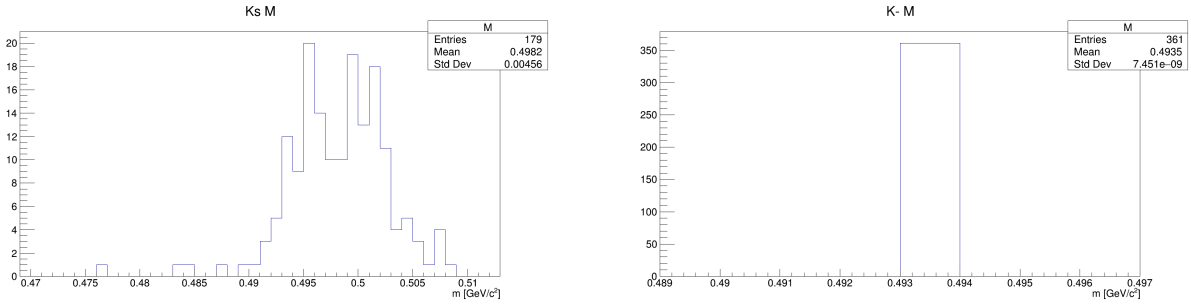
12. ábra. A kaonpárok invariáns tömegének diagramján 1.02 GeV-nél, a Φ-mezon

A jelben 2500 esemény volt, azaz 2500 db mezont generáltam. Nagyjából 50%-os eséllyel bomlottak el ezek kaon párokra, valamint a digitalizáció során, nagyjából 15%-os hatékony-

sággal tudott a program rekonstruálni, azaz nagyjából 180 db rekonstruált Φ-mezonra lehet számítani a *KFParticleFinder.root* fájlban. A fájlban ezzel ellentétben még csak 120 db-ot sikerült rekonstruálni, tehát a szimuláció további tökéletesítésre szorul.



Könnyű megtalálni azokat a részecskéket is amikké a Φ-mezonok elbomlottak. Így találhatunk a bomlástermékek között pionokat, kaonokat, K_S^0 részecskéket.



(a) K_S^0 részecskék

(b) Negatív kaonok.

A pionokat itt nem tüntettem fel külön, mivel egy ilyen folyamat során azok nem adnak informatív képet, lévén, hogy nem csak a Φ-mezon tud úgy bomlani, hogy pion is van a bomlástermékek között, de a kaonok és egyéb részecskék is, így a pionok multiplicitása igen nagy.

IV.5. Összegzés

A CBM detektor képes lesz arra, hogy felismerje és megtalálja a Φ-mezon bomlásokat, ezáltal a strange termelődést és a partonikus anyagot vizsgálni tudja. A szimulációk azt sugallják, hogy a detektor minden valószínűség szerint képes lesz detektálni a szükséges részecskéket a megfelelő hatásfokokkal.

V.. Nehézion fizika itthon

A Wigner Fizikai Kutatóközpontban dolgozó témavezetőmtől, Wolf Györgytől azt a feladatot kaptam, hogy az általa írt nehézion reakciós programhoz írjak egy klaszterező programot. Ez a szimuláció a korábban említett, PHSD és UrQMD modellekhez hasonló, hazai fejlesztésű projekt. A hadron-mag és mag-mag reakciókat transzport-egyenletek segítségével vizsgálva, a

BUU-modell[8] felhasználásával egy időfüggő, részecskék kölcsönhatását figyelembe vevő modell segítségével szimulálja ez a program.

Ennek kimenetén többek között szerepelhetnek bizonyos részecskék és azok momentum- és térbeli eloszlása. Detektortól függően máshogy lehet ezeket mérni. Ha olyan detektorunk van, ami csak töltött részecskéket mér, és a töltés nagyságát nem, akkor figyelembe kell vennünk, ha például térbeli (vagy impulzustérbeli) közelség miatt csak egy beütést kapunk. Így az én programom pontosan arra képes, hogy euklideszi-térben (vagy impulzustérben) klasztereket keres. Így a beütésszámra pontosabb jóslatot lehet majd adni tényleges detektor környezetben.

Továbbá a CBM kísérletben kutatni fogjak az egyszeres és kétszeres ritka magokat, amihez szintén szükséges a koaleszcencia szimulációja.

V.1. Az algoritmus

Nem tökéletesítettem még a programot, ha későbbiekben erre igény van természetesen fejlesztem. Egyenlőre hely- és impulzus-koordinátákat olvas be, majd ezután próbálja meg klaszterezni a részecskéket. A klaszterezéshez nem a legjobban ismert klaszterező algoritmust használtam hanem az úgynevezett minimális feszítő fa (vagy MST [9] a későbbiekben) algoritmust. Ez egy gráfban a lehető legrövidebb utat találja meg. Két pont akkor van összekötve a gráfban, ha egy adott minimum távolságnál közelebb vannak. Természetesen ez a minimális távolság is a bemenetről állítható. Egy részecske egy klaszter része, ha legalább az egyik részecskéhez a klaszterben kellően közel van.

Ennek az algoritmusnak talán az a legnagyobb előnye, hogy nem kell előre feltételezni, hogy hány klaszter van és azt sem, hogy azok vajon hol helyezkedhetnek el. Elméletben az algoritmus hatékonysága $O(\log m \cdot n)$ vagy $O(\log n \cdot n + m)$, ahol n a pontok száma a gráfban, míg m az élek száma. A hatékonyság a használt adatstruktúráktól függ. Ez természetesen Prim algoritmusára [10] igaz, vannak ennél hatékonyabb megoldások is, de számomra ez tűnt a legkényelmesebb, legmegvalósíthatóbb választásnak. Továbbá egy francia kutatócsoport Nantes-ban hasonló nehézion fizikai szimulációjában is ezt az algoritmust javasolják.

Az egyik elméleti nehézség a megvalósítás során az volt, hogy az algoritmus képes legyen több klasztert formálni. Hiszen miután nem tud továbbhaladni egy klaszterben, azaz nem tud több pontot hozzáadni, ki kell venni az adathalmazból a klaszterezett pontokat. Ezután lehet csak választani egy random pontot újra, és lefuttatni az eddigi algoritmust a már redukált gráfon.

V.2. A kód

A kódot mellékelem, ezután beszélek majd a bemenetéről és kimenetéről.

```
1#include <iostream>
2#include <vector>
3#include <utility>
4#include <set>
5#include <array>
6#include <string>
7#include <fstream>
```

[8] Boltzmann-Uehling-Uhlenbeck modell

[9] Minimal Spanning Tree

[10] `https://en.wikipedia.org/wiki/Prim\%27s_algorithm`

```cpp
#include <numeric> // for std::accumulate
#include <sstream> // for std::istringstream
#include <algorithm> // for std::for_each, std::find, etc.
#include <cmath> // for std::sqrt
#include <list>
#include <chrono> // to measure time

double infinity = 10000.; // handled as infinity in prim's algorithm
constexpr int dim = 6;
constexpr double scalingFactor = 13.7; // for distance calculations

struct DataPoints{

        std::vector<int> hadron;
        std::vector<int> charge;
        std::vector<double> mass;
        std::vector< std::array< double, dim > > posAndMom;
        std::vector<int> num_of_coll;

};

std::istream& operator>>(std::istream& is, DataPoints& points);

double distance( const std::array<double, dim>& a, const std::array<double, dim
    >& b);

bool prim(std::ostream& fout,
          std::vector<bool>& visitedVertices,
          std::vector< int >& vertexAncestors,
          std::vector<std::pair<int, int>>& edges,
          std::vector<std::vector<std::pair<int, double>>>& Graph,
          std::list<int>& unclusteredVertices);

int main(int argc, char* argv[]){

    std::ifstream fin(argv[1]);
    infinity = atof(argv[2]);

    DataPoints data;
    fin >> data;

    int numberOfVertices = data.mass.size();

    std::vector<std::vector<std::pair<int, double>>> Graph(numberOfVertices);
    std::vector<bool> visitedVertices(numberOfVertices, false);
    std::vector<int> vertexAncestors(numberOfVertices, -1);
    std::vector<std::pair<int,int>> edges;

    int P,Q;
    double distanceBetweenVertices;

    auto t0 = std::chrono::high_resolution_clock::now();

    for( unsigned int i = 0; i < numberOfVertices; i++ ){

        P = i;
```

```
63
64          for( unsigned int j = 0; j < i; j++ ){
65
66              Q = j;
67
68              distanceBetweenVertices = distance( data.posAndMom[i],data.posAndMom
    [j] );
69
70              if(distanceBetweenVertices < infinity){
71
72                  Graph[P].push_back(std::make_pair(Q, distanceBetweenVertices));
73
74                  Graph[Q].push_back(std::make_pair(P, distanceBetweenVertices));
75
76              }
77          }
78      }
79
80      auto t1 = std::chrono::high_resolution_clock::now();
81
82      std::cout << "Graph construction took: " <<
83      std::chrono::duration_cast< std::chrono::microseconds >(t1-t0).count() << "
    microseconds\n";
84
85      std::list<int> unclusteredVertices;
86      for(unsigned int i = 0; i < numberOfVertices; i++){
87          unclusteredVertices.push_back(i);
88      }
89
90      std::ofstream fout(argv[3]);
91
92      auto t2 = std::chrono::high_resolution_clock::now();
93
94      while(!prim(fout, visitedVertices, vertexAncestors, edges, Graph,
    unclusteredVertices) && unclusteredVertices.size()!=0);
95
96      auto t3 = std::chrono::high_resolution_clock::now();
97
98      std::cout << "Clustering took: " <<
99      std::chrono::duration_cast< std::chrono::microseconds >(t3-t2).count() << "
    microseconds\n";
100
101     return 0;
102
103 }
104
105 std::istream& operator>>(std::istream& is, DataPoints& points){
106
107     std::array<double, dim> tempPosAndMom;
108
109     std::string line;
110
111     while(std::getline(is, line)){
112
113         double tempData;
114
```

```
115         std::istringstream rawData(line);
116 
117         rawData >> tempData;
118         points.hadron.push_back(int(tempData));
119         rawData >> tempData;
120         points.charge.push_back(int(tempData));
121         rawData >> tempData;
122         points.mass.push_back(tempData);
123 
124         for(unsigned int i=0; i < dim; i++){
125             rawData >> tempPosAndMom[i];
126         }
127 
128         points.posAndMom.push_back(tempPosAndMom);
129 
130         rawData >> tempData;
131         points.num_of_coll.push_back(int(tempData));
132 
133     }
134 
135     return is;
136 
137 }
138 
139 double distance( const std::array<double, dim>& a, const std::array<double, dim
    >& b){
140 
141     double distPos = 0.0;
142     double distMom = 0.0;
143 
144     for(unsigned int i = 0; i < dim/2; i++){
145 
146         distPos += (a[i] - b[i])*(a[i] - b[i]);
147         distMom += scalingFactor*(a[dim/2+i] - b[dim/2+i])*(a[dim/2+i] - b[dim
    /2+i]);
148 
149 
150     }
151 
152     return std::sqrt(distPos+distMom);
153 
154 }
155 
156 bool prim(std::ostream& fout,
157           std::vector<bool>& visitedVertices,
158           std::vector< int >& vertexAncestors,
159           std::vector<std::pair<int, int>>& edges,
160           std::vector<std::vector<std::pair<int, double>>>& Graph,
161           std::list<int>& unclusteredVertices){
162 
163     if(unclusteredVertices.size()==0){
164         return true;
165     }
166 
167     int numberOfVertices = visitedVertices.size();
168     std::list<int> unclusteredVerticesBefore = unclusteredVertices;
```

```
169
170     int statingVertex;
171
172     statingVertex = unclusteredVertices.front();
173     unclusteredVertices.remove(statingVertex);
174
175     std::vector<double> distanceSet(Graph.size(), infinity);
176
177     std::set<std::pair<double, int>> vertexSet;
178     for(unsigned int i = 0; i < numberOfVertices; i++){
179         vertexSet.insert(std::make_pair(distanceSet[i], i));
180     }
181     vertexSet.erase(vertexSet.find(std::make_pair(distanceSet[statingVertex],
    statingVertex)));
182     distanceSet[statingVertex] = 0;
183     vertexSet.insert(std::make_pair(distanceSet[statingVertex],statingVertex));
184
185     while(!vertexSet.empty()){
186
187         std::pair<double, int> topVertex = *vertexSet.begin();
188         vertexSet.erase(vertexSet.begin());
189
190         int nextVertex = topVertex.second;
191
192         if(topVertex.first == infinity) break;
193
194         unclusteredVertices.remove(nextVertex);
195
196         visitedVertices[nextVertex]  = true;
197
198         if(nextVertex != statingVertex){
199             edges.push_back(std::make_pair(vertexAncestors[nextVertex],
    nextVertex));
200         }
201
202         for(unsigned int i = 0; i < Graph[nextVertex].size(); i++){
203             if(visitedVertices[Graph[nextVertex][i].first]==false){
204                 int updatedNextVertex = Graph[nextVertex][i].first;
205                 double edgeWeight = Graph[nextVertex][i].second;
206                 vertexSet.erase(vertexSet.find(std::make_pair(distanceSet[
    updatedNextVertex], updatedNextVertex)));
207                 distanceSet[updatedNextVertex] = edgeWeight;
208                 vertexSet.insert(std::make_pair(distanceSet[updatedNextVertex],
    updatedNextVertex));
209                 vertexAncestors[updatedNextVertex] = nextVertex;
210             }
211         }
212     }
213
214     std::list<int> clusteredVertices;
215     std::set_difference(unclusteredVerticesBefore.begin(),
    unclusteredVerticesBefore.end(),
216                         unclusteredVertices.begin(), unclusteredVertices.end(),
217                         std::back_inserter(clusteredVertices));
218     fout << clusteredVertices.size() << "\n";
219
```

```
220     edges.clear();
221 
222     int pointsToCluster = 0;
223     for(unsigned int i = 0; i < numberOfVertices; i++){
224     if(visitedVertices[i] == true){
225         Graph[i].clear();
226         }else{
227             pointsToCluster++;
228         }
229     }
230 
231     return pointsToCluster == 0 ? true : false;
232 
233 }
```

V.21. Bemeneti paraméterek

A program parancssorról működik. Három paramétert vár a futás során. Az első paraméter a bemeneti fájl neve, a második a maximális klaszeterezési távolság, azaz a maximálisan definiálható élhossz két pont között. A harmadik paraméter pedig a kimeneti fájl neve, ami egyesével listázza majd a klaszeterek méreteit és az összes klaszterek számát. A program képernyőre írja a gráf elkészítésének idejét és a klaszterezés idejét is ms-ban. Ezt parancssorról át lehet irányítani egy tetszőleges fájlba (» time.dat).

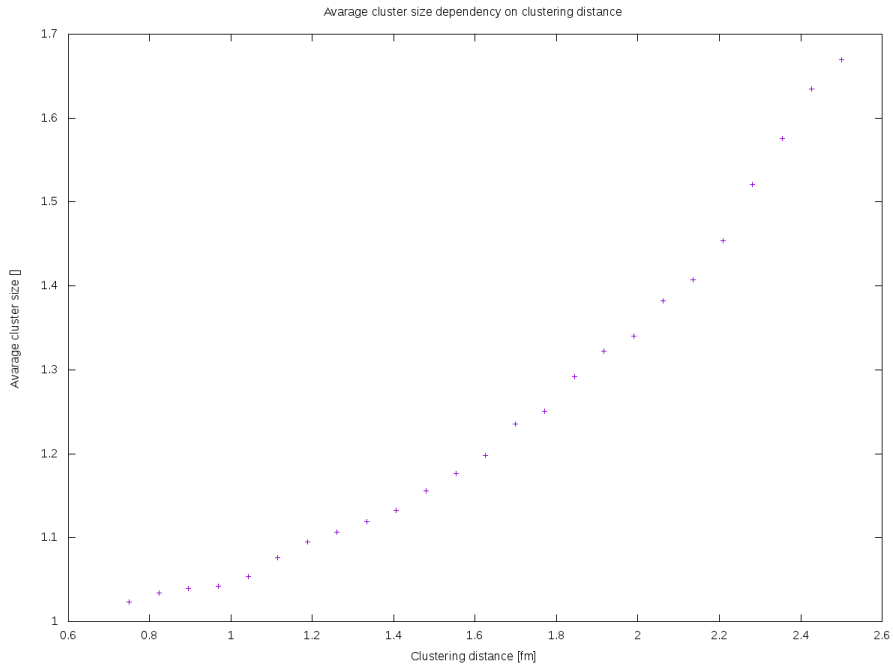
A bemeneti fájl formátuma adott. Ebben részecske adatok szerepelnek, és adott időlépésenként haladva követjük végig a részecskéket. Sorénként a következő adatokat kapjuk: nukleon (igen/nem), töltés, tömeg(GeV), px, py, pz (GeV/c), rx, ry, rz (fm), ütközések száma.

```
1  1  0.938300 −0.177627  0.025891 −0.077826 −7.935590  0.056308 −11.041818  1
1  1  0.938300  0.418485  0.316837  0.882886 12.817832  5.254027  6.157795  2
1  1  0.938300 −0.247703  0.057946 −0.044372 −12.626606  4.831421 −19.696069  1
1  1  0.938300 −0.022614 −0.215107  0.223749 −4.957242 −6.073233 −4.840523  2
1  1  0.938300 −0.208557 −0.300033  0.080906 −5.851900 −14.546136 −11.728940  4
1  0  0.938300 −0.556290  0.208243  0.389433 −16.943628  12.829147 −3.252291  7
1  1  0.938300 −0.045176 −0.188415  0.277124 −3.621005 −9.789178 −6.976719  3
1  1  0.938300 −0.562875 −0.475173  0.274616 −17.897499 −17.592449 −6.268526  3
1  1  0.938300 −0.032136 −0.021516 −0.007764 −5.412685  0.558902 −15.674200  1
```

Három szimulációs fájlt kaptam, mindegyik más időpillanatban állt meg. Az első 40 fm/c után, a második 50 fm/c, a harmadik pedig 60 fm/c idő után. A fájlok 100 eseményt tárolnak, mindegyikben 394 db nukleon szerepel. Az ütközések 3 fm-es impakt paraméterrel játszódtak le. A részecskék távolságának definícióját még tökéletesíteni kell. Először is elvégeztem egy klaszterezést külön térben és impulzus térben:

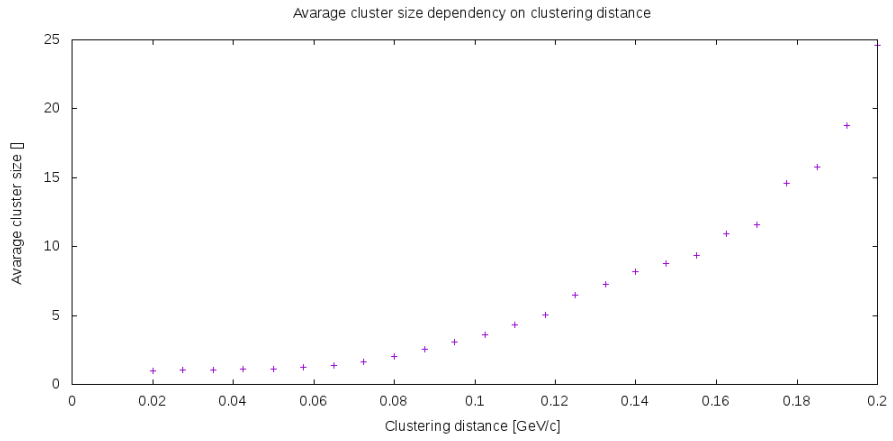
V.3. Távolság függés

Valamelyik eseményre elvégezve a klaszterezési távolságtól való függés vizsgálatát a következőket kaptam.



14. ábra. Térbeli klaszterezés eredménye. Klaszterezési távolság - átlagos klaszterméret

A számolást természetesen térbeli, és impulzustérbeli klaszterezésre is elvégeztem.



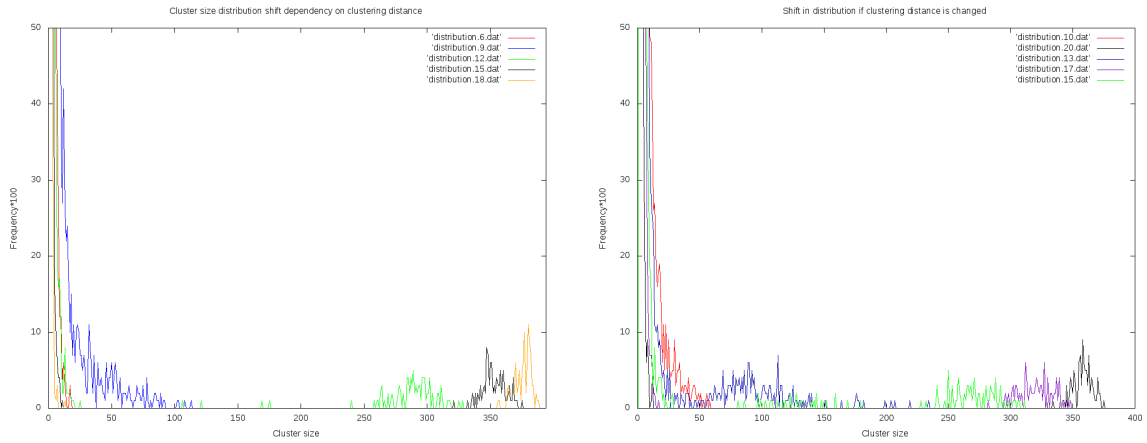
15. ábra. Impulzustérbeli klaszterezés eredménye. Klaszterezési távolság - maximális klaszterméret

Térben a klaszterezés életképes, ha a távolság a részecskék között 1 fm alatt van, míg impulzustérben ez az érték 0.08 GeV/c alatti. Így jól látható, hogy az impulzustérbeli távolság nagyjából egy nagyságrenddel kisebb, mint a térbeli távolság. Az előbbi klaszterezések még a kód korábbi verziójával készültek, ez megtalálható a GitHub profilom alatt [11]. Ahhoz, hogy ténylegesen összeálljanak a részecskék az kell, hogy mind térben, mind impulzustérben közel legyenek egymáshoz. Ehhez felvettem egy olyan vektort aminek az első 3 komponense a térbeli koordinátákat tartalmazza, második három komponenese az impulzustérbeli koordinátákat és

[11] github link

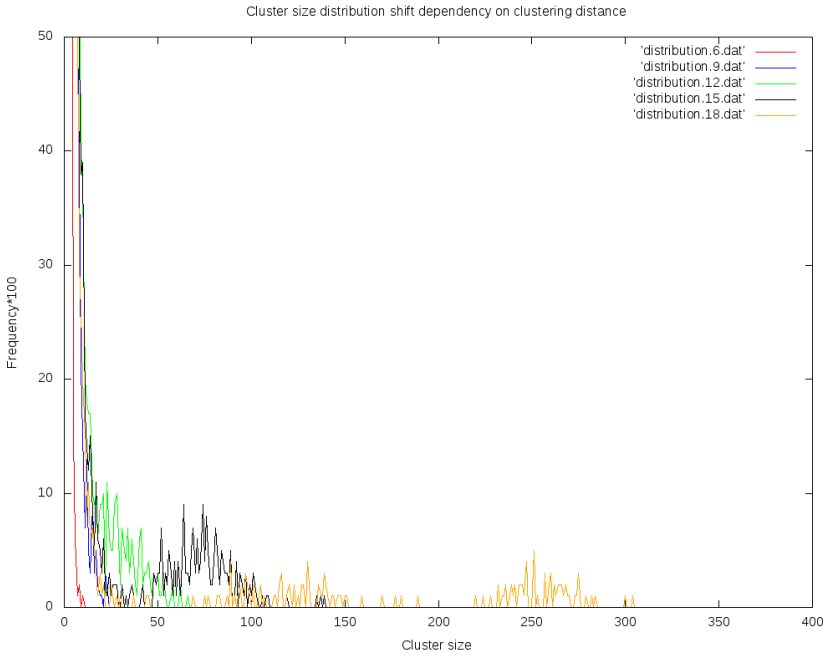
ezeket beszoroztam egy $scalingFactor = 13.7$-ral, hogy kompenzáljak a nagyságrendi különbség miatt. Ez még nagyon kezdetleges és további finomításra szorul. Ezután különböző klaszterezési távolságkora, amelyek lineárisan nőttek lefuttattam a klaszterezést a 100 adatsorra és ebből készítettem eloszlás hiszrogrammokat. A különböző színek különböző klaszterező távolságokat jelölnek ugyanazon a mintán.

Az klaszterméret eloszlások változása a klaszterező távolság növekedésével rendere 40, 50, 60 fm/c után:



(a) Klaszterszám - klaszterméret eloszlás 40 fm/c idő után

(b) Klaszterszám - klaszterméret eloszlás 50 fm/c idő után



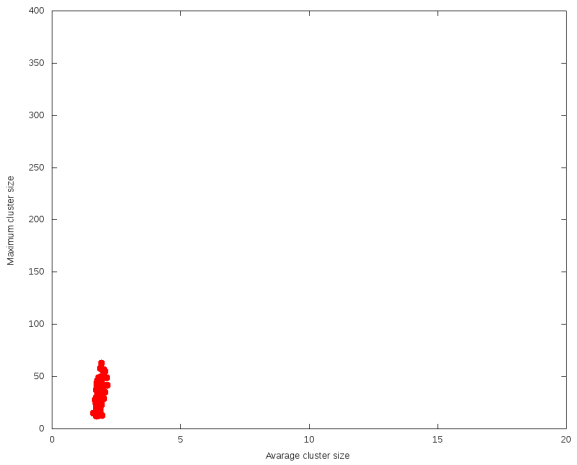
17. ábra. Klaszterszám - klaszterméret eloszlás 60 fm/c idő után

Jól megfigyelhető, ahogy a klaszter távolság növelésével egyre gyakoriabbak lesznek a nagyméretű klaszterek. Ezek jelenthetnek kisebb/nagyobb magokat. Jól látható az is, hogy minél

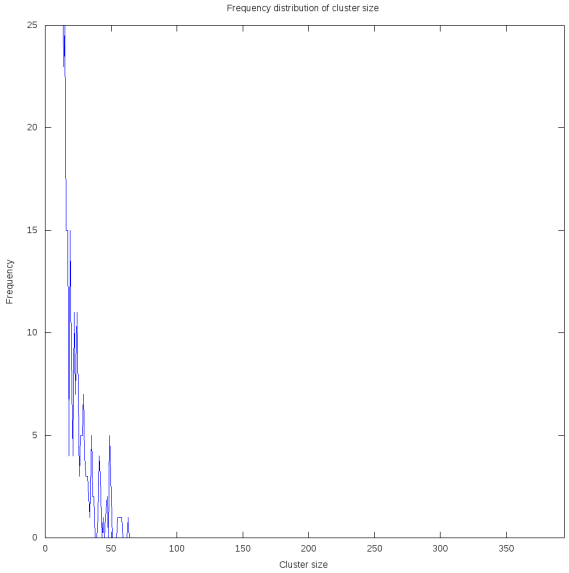
tovább futott a program, a klaszeretek átlagos mérete egyre csökkent, azaz távolodtak egymástól a részecskék Szemléletes, például, hogy kis impakt paraméternél nagy számú 'törmelék' keletkezik, hiszen a 1/2 nukleonos klaszterek száma végig magas.

A továbbiakban vizsgáltam, hogy mi történik a klaszterező méret növelésével. Itt azt néztem meg, hogy az adott 100 párhuzamos fájlra nagyjából hasonló eredményeket kapok-e a klaszterezés után. Ezt úgy állapítottam meg, hogy felvettem egy diagramot aminek vízszintes tengelyén az átlagos klaszterméret, függőleges tengelyén pedig a maximum klaszter méret szerepelt mindegyik fájlra. Ez 100 pontot jelent a diagrammon. Ha ezt ábrázolom akkor hasonló eredmény az, ha adott pontcsoportok nagyjából egy helyre koncentrálódnak. ne szórjanak nagyon, hiszen ezek részecske csoportosulásokat jelentenek. Elnyújtott formájuk amiatt lehet, mivel az 1/2 részecskés klaszterek száma nagyon nagy. Ennél pontosabb definíciót nem fogok adni, mivel a távolság számítása ebben kulcsfontossága és azon még javítanom kell a későbbiekben. Különböző, egyre növekedő klaszterező távolságokra az eredeményeim a következőek.

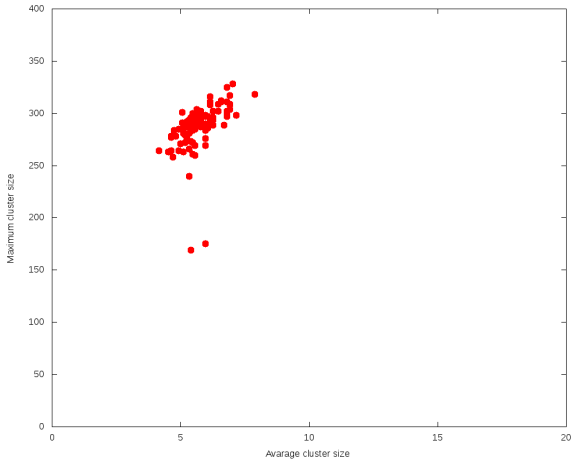
40 fm/c esetén:



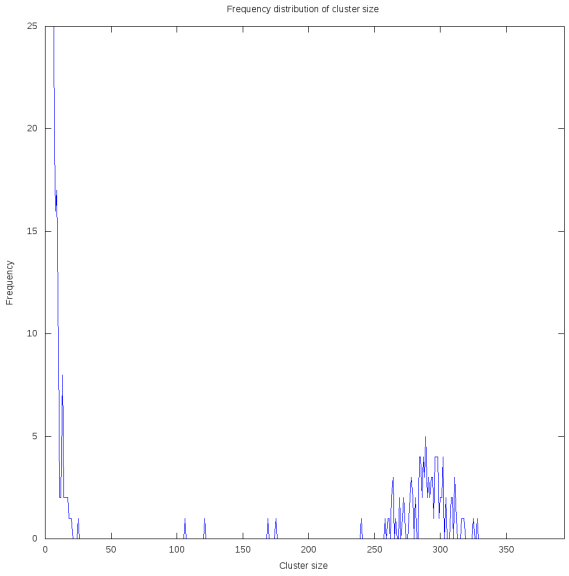
(a) Átlagos klaszterméret - Maxmimális klaszterméret diagram



(b) Klaszterméret eloszlás 100 eseményből

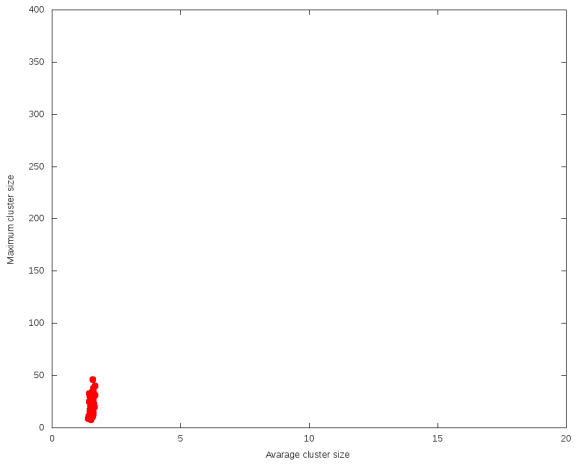


(a) Átlagos klaszterméret - Maxmimális klaszterméret diagram

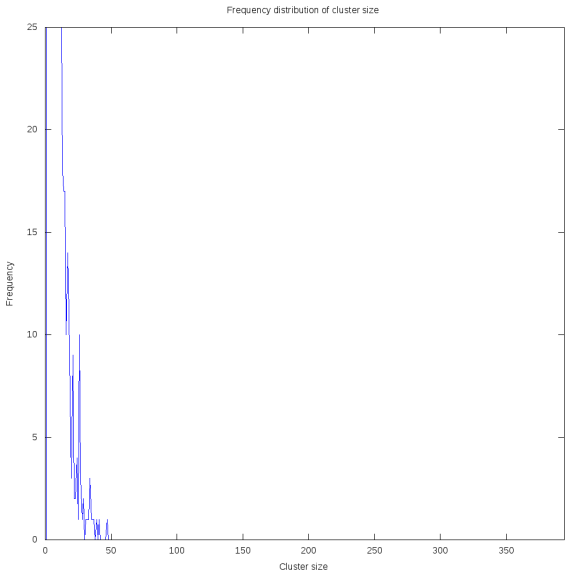


(b) Klaszterméret eloszlás 100 eseményből

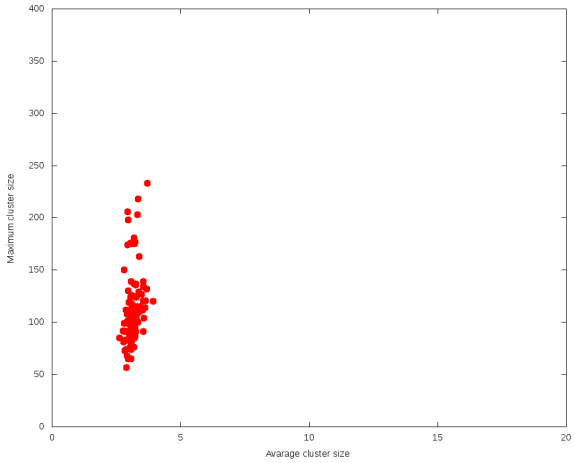
Látható az eloszlásban történő eltolódás és az is, hogy az események nagyjából hasonlóak. 50 fm/c esetén:



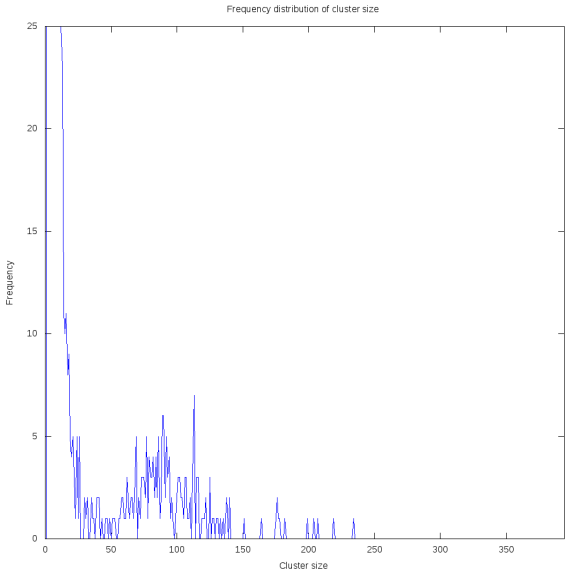
(a) Átlagos klaszterméret - Maxmimális klaszterméret diagram



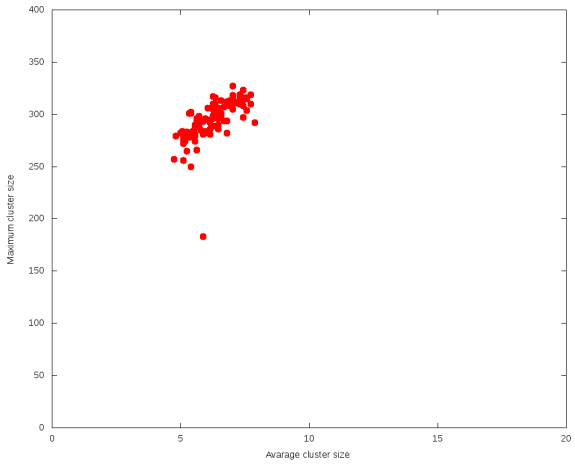
(b) Klaszterméret eloszlás 100 eseményből



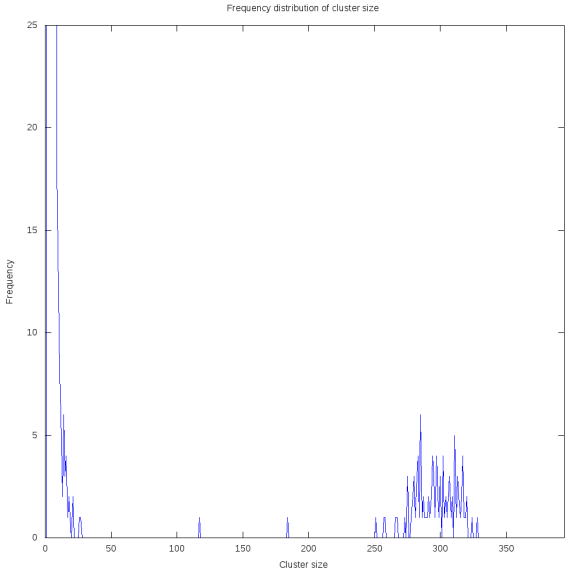
(a) Átlagos klaszterméret - Maxmimális klaszterméret diagram



(b) Klaszterméret eloszlás 100 eseményből



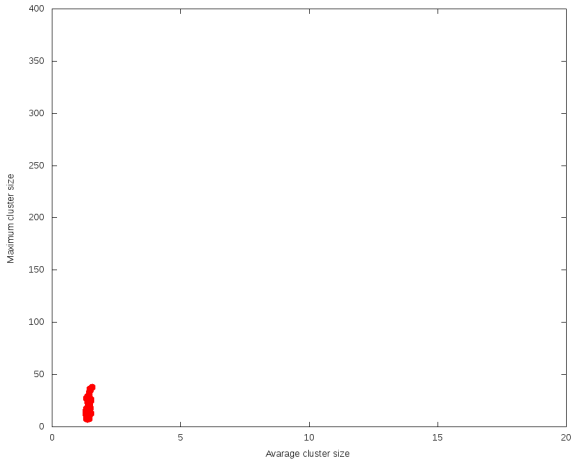
(a) Átlagos klaszterméret - Maxmimális klaszterméret diagram



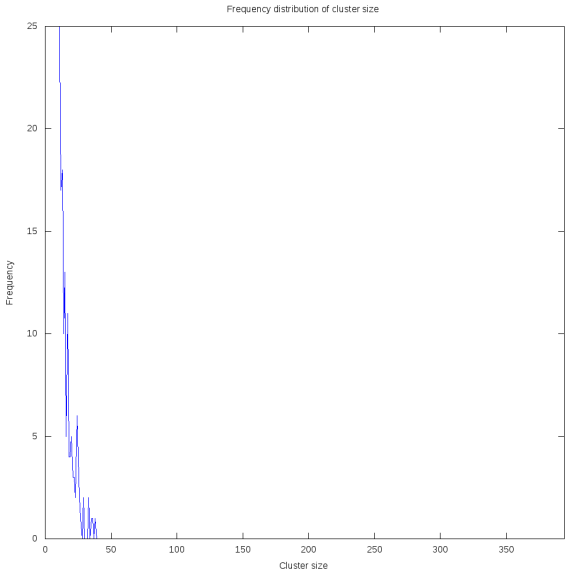
(b) Klaszterméret eloszlás 100 eseményből

Itt már a második diagramon jól látszik, hogy kezd szétválni a ponthalmaz két részre. A következő esetben már ez sokkal jobban látszik majd.

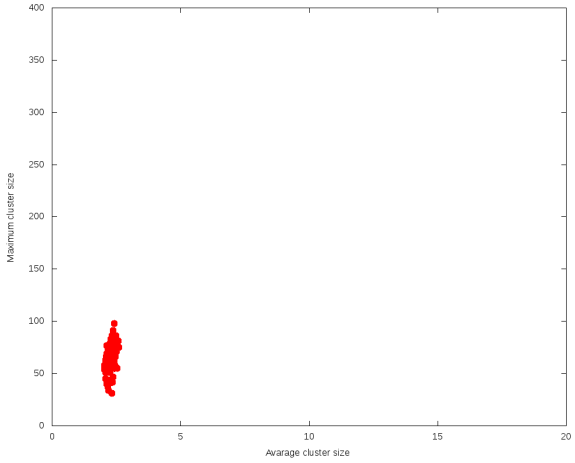
60 fm/c esetén:



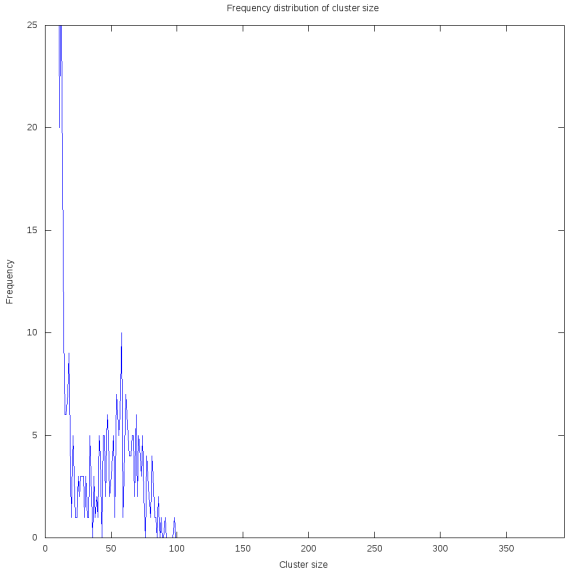
(a) Átlagos klaszterméret - Maxmimális klaszterméret diagram



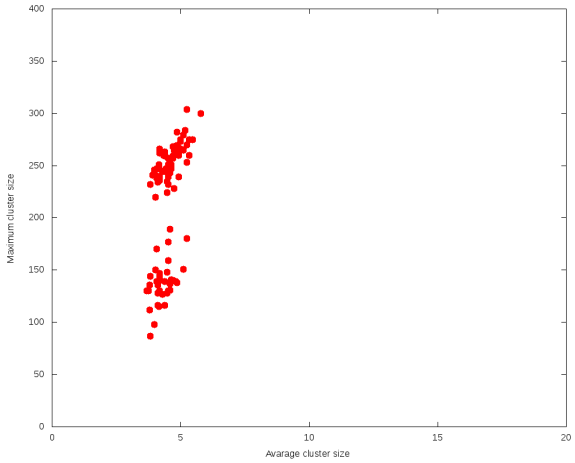
(b) Klaszterméret eloszlás 100 eseményből



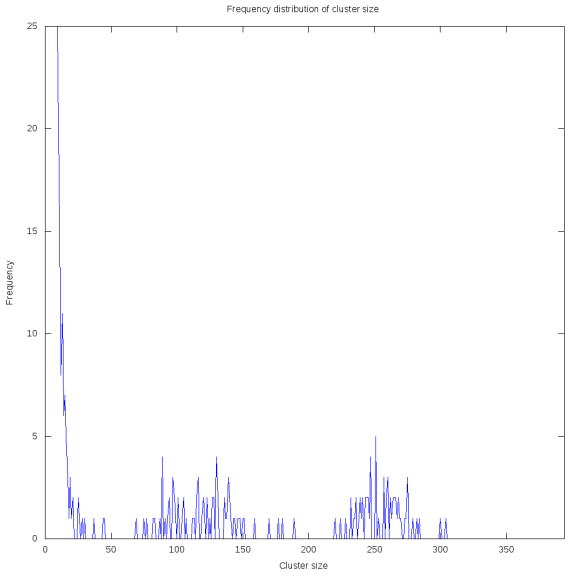
(a) Átlagos klaszterméret - Maxmimális klaszterméret diagram



(b) Klaszterméret eloszlás 100 eseményből



(a) Átlagos klaszterméret - Maxmimális klaszterméret diagram



(b) Klaszterméret eloszlás 100 eseményből

Az utolsó (a) ábrán jól látszik, hogy a ponthalmazok jól szemléltetik az eloszlás diagrammot, amint az két jól elkülöníthető részre esik szét.

V.4. Kimenet

A kimeneten tehát egy olyan fájlt kapunk, amiből meghatározható igen könnyen a maximális klaszterméret és az átlagos klaszterméret is. Az fentebbi ábrákat is ezek alapján készítettem egy egyszerű programmal. Ezek mellett még külön fájlokban megkapjuk a klaszterek elemeit is a program korábbi verziójában.

V.5. Sebesség, konklúzió

A kód sebessége nagyban függ a definiált maximum távolságtól. Ha az előbbi algoritmust egy igen nagy szám, akkor egy nagy klasztert találunk, viszonylag gyorsan, hiszen az MST keresés hiba nélkül lefut. Azonban minél nagyobb távolságokra is éleket rakunk a kreált gráfba annál több számítást kell végeznünk és a sebesség nagyban leromlik. Lényegében mondható az, hogy a klaszterezés belátható időn belül lefut tetszőlegesen nagy adathalmazra, hacsak a rendszer ki nem fogy a memóriából. 12 ezer sornyi adat klaszterezése esetén ez már megtörténhet. A program több dologra is képes mint amire jelenleg használva van. Az éleket tudja definiálni és menteni is, ez az MST algoritmus sajátossága, de nekem nincs rá szükségem ebben az alkalmazásban. A távolság számítást módosítanom kell majd, hiszen jelenleg csak euklideszi normám van, ami nem a legmegfelelőbb, de ez csak egy belső függvény átírást igényel majd. A kommentált kódrészletek a kimeneti fájlok emberi olvashatóságáért feleltek volna részben, valamint a további funkcionalitást kapcsoltam ki, de az adatok feldolgozása során kényelmesebb volt így eljárnom.

Hivatkozások

[1] Friman, Höhne, Knoll, Leupold, Randrup, Rapp, Senger
The CBM Physics Book - Compressed Baryonic Matter in Laboratory Experiments 2011 (Springer) Lect. Notes Phys.

[2] Tapia Takaki, J.D.
ALICE Collaboration Journal of Physics G Nuclear Physics, 35, 044058 2008

[3] V.Vovchenko, I.Vassiliev, I.Kisel, M.Zyzak
Φ-meson production in Au+Au collisions and its feasibility in the CBM experiment, CBM Progress Report 2014

[4] Bravina, L., Csernai, L., Faessler, A., et al. 2003, Nuclear Physics A, 715, 665

[5] F. Wang, R. Bossingham, Q. Li, I. Sakrejda, N. Xu
Φ-meson reconstruction in the STAR TPC, 1998

[6] Hans Rudolf Schmidt
Hyperons at CBM-FAIR, Journal of Physics: Conference Series 736

[7] The European Physics Journal A
Challenges in QCD matter physics - The scientific programme of the Compressed Baryonic Matter experiment at FAIR

VI.. Köszenetnyílvánítás

Ezúton is szeretném megköszönni témevezetőmnek, Wolf Györgynek a sok segítséget és lehetőséget amit nyújtott nekem.