

Estudi d'Alternatives per la Reconstrucció de Traces del VELO d'LHCb

per

Núria Valls Canudas

Un treball de final de carrera presentat en satisfacció
parcial dels requisits per al grau de

Enginyeria de Sistemes de Telecomunicació

de la

Universitat Ramon Llull, La Salle BCN

Maig 2018

Resum

Estudi d'Alternatives per la Reconstrucció de Traces del VELO d'LHCb

per

Núria Valls Canudas

Enginyeria de Sistemes de Telecomunicació

Universitat Ramon Llull, La Salle BCN

Professor Xavier Vilasís Cardona, Doctor

El projecte d'LHCb del CERN realitzarà una millora del maquinari i programari de l'experiment el pròxim any 2019, amb l'objectiu de millorar el rendiment dels estudis que s'hi realitzen. Un dels principals reptes d'aquesta actualització és implementar un algoritme capaç de reconstruir les traces que deixen les partícules dins dels detectors de l'experiment en un temps de l'ordre de mil·lisegons.

Amb aquesta estricta condició de temps, en aquest treball es realitza un estudi de diferents mètodes que puguin ser aplicats per la reconstrucció de traces del detector VELO d'LHCb. Es realitza també, l'anàlisi dels resultats obtinguts en execucions sobre dades reals amb l'objectiu de proporcionar una optimització del temps d'execució d'un algoritme ja implementat amb aquest propòsit mitjançant estructures de dades no lineals.

Paraules clau

Reconstrucció de traces; VELO; LHC; LHCb *upgrade*; transformada de Hough; autòmat cel·lular; *R-Tree*; temps d'execució; optimització.

Dedicat a aquelles persones que han dipositat tota la seva confiança en mi i en la meva
capacitat personal i de superació.

Índex

Índex	ii
Índex de figures	iv
Índex de taules	viii
1 Introducció	1
1.1 L'LHC al CERN	1
1.2 Cronologia dels Experiments	3
1.3 L'experiment LHCb	4
1.4 Condicions de Funcionament	6
2 Detector VELO	9
2.1 Descripció General	9
2.2 Disseny de l'Actualització	9
2.3 Classificació de Traces	12
2.4 VELO Tracking	13
3 Descripció de les Dades Reals	15
3.1 Simulació Monte Carlo	15
3.2 <i>Event Model</i>	16
4 Mètode Clàssic: Transformada de Hough	19
4.1 Principi de l'Algorisme	19
4.2 Primera Implementació	21
4.3 Adaptació per un Sistema Tridimensional	23
4.4 Adaptació a les Dades Reals	25
5 Autòmat Cel·lular	29
5.1 Introducció i Principi	29
5.2 Primera Implementació	31
5.3 Adaptació a les Dades Reals	37
5.4 Resultats	40

5.5	Temps d'Execució	43
6	Procés d'Optimització de l'Algoritme	47
6.1	Introducció	47
6.2	Introducció a l' <i>R-Tree</i>	48
6.3	Aplicació de l' <i>R-Tree</i> a l'Algorisme de Reconstrucció	50
6.4	Taules de <i>Hash</i>	52
6.5	Aplicació de taules de <i>Hash</i> a l'Algorisme de Reconstrucció	53
6.6	Resultats	54
6.7	Mesura de Temps	57
6.8	Optimització de Paràmetres	58
6.9	Comparació de Resultats	64
7	Conclusions	69
A	Gràfiques d'optimització de l'<i>R-Tree</i>	71
	Bibliografia	77

Índex de figures

1.1	Esquema de la cadena d'acceleradors del CERN	2
1.2	Cronologia de l'experimentació d'LHC	3
1.3	Vista lateral del detector LHCb.	4
1.4	Diagrames del <i>trigger</i> , a l'esquerra corresponent al <i>Run 1</i> i a la dreta corresponent al <i>Run 2</i>	7
1.5	Diagrama del <i>trigger</i> corresponent al <i>Run 3</i>	8
2.1	Disseny esquemàtic del VeloPix.	10
2.2	Representació gràfica de l'àrea d'acceptació dins del VeloPix.	11
2.3	Representació del tipus de trases sobre el detector.	12
3.1	Diagrama del procés de generació d'una simulació Monte Carlo d'LHC.	16
3.2	Diagrama de la classe event	17
3.3	A l'esquerra, diagrama de la classe hit , a la dreta, diagrama de la classe sensor	17
3.4	Diagrama de la classe track	18
4.1	imatge extreta del <i>Big European Bubble Chamber</i> , iniciat al CERN l'any 1973[21].	20
4.2	Procés d'extracció dels paràmetres de totes les possibles rectes amb diferents angles que passen pels diferents punts d'un conjunt en una imatge.	21
4.3	Conjunt de punts alineats en tres rectes sobre el pla x, y	22
4.4	Espai de Hough resultant de la transformada sobre la imatge de la figura 4.3.	22
4.5	Conjunt de punts alineats en dues rectes tridimensionals.	23
4.6	Projeccions de la imatge de la figura tridimensional 4.5 sobre el pla XZ a l'esquerra, i sobre el pla YZ a la dreta.	24
4.7	Conjunt de punts alineats en dues rectes tridimensionals.	24
4.8	A l'esquerra, projecció sobre els eixos XZ d'un esdeveniment d'LHCb al sub-detector VELO. A la dreta, projecció sobre els eixos YZ	26
4.9	Espai de Hough resultant d'aplicar la transformada sobre la projecció XZ	26
4.10	A l'esquerra, projecció dels <i>hits</i> d'un esdeveniment sobre els eixos XY , a la dreta, espai de Hough generat aplicant la transformada sobre la imatge de l'esquerra.	27
5.1	Closca del cargol de mar <i>Conus textile</i> amb estructures fractals caracteritzables per un autòmat cellular.	30

5.2	Graella d'un A.C. unidimensional de 10 cèl·lules.	31
5.3	Graella d'un A.C. unidimensional de 10 cèl·lules amb estats inicials aleatoris.	31
5.4	Selecció dels veïns de la setena cèl·lula d'un A.C. unidimensional.	31
5.5	Gràfic de la funció de transició utilitzada.	32
5.6	Gràfic de l'evolució d'un autòmat cel·lular de 100 cèl·lules al llarg de 50 generacions.	33
5.7	Gràfic de l'evolució d'un autòmat cel·lular de 100 cèl·lules al llarg de 50 generacions amb la regla 190.	33
5.8	Gràfic de l'evolució d'un autòmat cel·lular de 100 cèl·lules al llarg de 50 generacions amb la regla 30.	34
5.9	Graella d'un autòmat cel·lular bidimensional amb dos estats, ressaltats els veïns de la cèl·lula (7, 6).	35
5.10	A l'esquerra, procés de <i>mort</i> d'una cèl·lula a causa de subpoblació (un únic veí viu). A la dreta, procés de <i>naixement</i> d'una cèl·lula a causa de reproducció (exactament tres veïns vius).	36
5.11	A l'esquerra, tercera generació del joc de la vida amb unes condicions inicials aleatòries. A la dreta, un patró cíclic de tres períodes batejat amb el nom de <i>Pulsar</i>	36
5.12	Representació del procés de cerca de veïns amb les dades del detector VELO. Les parelles de punts unides per una línia vermella, segons la norma, no seran mai veïns. Les parelles de punts unides per una línia verda, són susceptibles a formar una recta.	38
5.13	Gràfica de l'eficiència de l'algoritme de reconstrucció en funció del nombre de <i>hits</i> per esdeveniment.	41
5.14	Gràfica del nombre de <i>clone tracks</i> reconstruïdes en funció del nombre de <i>hits</i> per esdeveniment.	42
5.15	Gràfica de la taxa de <i>ghost tracks</i> reconstruïdes en funció del nombre de <i>hits</i> per esdeveniment.	42
5.16	Gràfica del temps d'execució de l'algoritme, sobre el <i>hardware</i> especificat, en funció del nombre de <i>hits</i> dels diferents esdeveniments.	44
6.1	Projecció sobre el pla <i>XY</i> dels <i>hits</i> dels sensors 48, 49, 50 i 51 del detector VELO.	48
6.2	Arbre simple sense ordenar.	49
6.3	Exemple de l'estruatura d'un <i>R-Tree</i> i representació de l'organització de cadascun dels seus nodes.	49
6.4	Esquema de la projecció d'un <i>hit</i> situat al mòdul z_{origen} , en color ver, sobre el mòdul $z_{origen} + 1$, per trobar els seus <i>hits</i> més pròxims del mateix mòdul, marcats en groc.	52
6.5	Esquema de funcionament d'una taula de <i>Hash</i>	53
6.6	Gràfica de l'eficiència de l'algoritme de reconstrucció optimitzat en funció del nombre de <i>hits</i> per esdeveniment.	55
6.7	Gràfica de la taxa de <i>clone tracks</i> de l'algoritme de reconstrucció optimitzat en funció del nombre de <i>hits</i> per esdeveniment.	56

6.8	Gràfica de la taxa de <i>ghost tracks</i> de l'algoritme de reconstrucció optimitzat en funció del nombre de <i>hits</i> per esdeveniment.	56
6.9	Gràfica del temps d'execució de l'algoritme optimitzat en funció del nombre de <i>hits</i> de cada esdeveniment.	57
6.10	Gràfic tridimensional d'eficiència, <i>clone rate</i> i <i>ghost rate</i> en funció del nombre de veïns i el nombre de <i>hits</i> per esdeveniment, vist des de la perspectiva del nombre de veïns.	59
6.11	Gràfic tridimensional d'eficiència i temps d'execució en funció del nombre de veïns i el nombre de <i>hits</i> per esdeveniment.	59
6.12	Gràfic tridimensional de la suma dels valors normalitzats d'eficiència, <i>clone rate</i> , <i>ghost rate</i> i temps d'execució en funció del nombre de veïns i el nombre de <i>hits</i> per esdeveniment, projectat sobre el nombre de veïns.	60
6.13	Gràfic tridimensional d'eficiència, <i>clone rate</i> i <i>ghost rate</i> en funció dels valors de tolerància i el nombre de <i>hits</i> per esdeveniment, vist sobre l'eix de tolerància.	61
6.14	Gràfic tridimensional d'eficiència i temps d'execució en funció de la tolerància i el nombre de <i>hits</i> per esdeveniment, amb cerques de dos veïns.	62
6.15	Gràfic tridimensional d'eficiència, <i>clone rate</i> i <i>ghost rate</i> en funció dels valors de tolerància i el nombre de <i>hits</i> per esdeveniment, vist sobre l'eix de tolerància i superposant els resultats de diferents nombres de veïns.	63
6.16	Diagrama de caixes de la variació de la suma de valors normalitzats en funció del nombre de <i>hits</i> de cada esdeveniment per a 2, 3 i 4 veïns.	64
6.17	Gràfic comparatiu d'eficiència en funció del nombre de <i>hits</i> per esdeveniment entre l'algoritme original i l'optimitzat.	65
6.18	Diagrama de caixes de la variació d'eficiència en funció del nombre de <i>hits</i> per esdeveniment de l'algoritme original i l'optimitzat. La mitjana de la distribució es marca amb un triangle verd i la mediana amb una línia groga.	65
6.19	Gràfic comparatiu de <i>clone tracks</i> en funció del nombre de <i>hits</i> per esdeveniment entre l'algoritme original i l'optimitzat.	66
6.20	Gràfic comparatiu de <i>ghost tracks</i> en funció del nombre de <i>hits</i> per esdeveniment entre l'algoritme original i l'optimitzat.	67
6.21	Gràfic comparatiu del temps d'execució en funció del nombre de <i>hits</i> per esdeveniment entre l'algoritme original i l'optimitzat.	68
A.1	Gràfic tridimensional d'eficiència, <i>clone rate</i> i <i>ghost rate</i> en funció del nombre de veïns i el nombre de <i>hits</i> per esdeveniment.	71
A.2	Gràfic tridimensional de la suma dels valors normalitzats d'eficiència, <i>clone rate</i> , <i>ghost rate</i> i temps d'execució en funció del nombre de veïns i el nombre de <i>hits</i> per esdeveniment.	72
A.3	Gràfic tridimensional d'eficiència, <i>clone rate</i> i <i>ghost rate</i> en funció de la tolerància i el nombre de <i>hits</i> per esdeveniment amb cerques de dos veïns.	72
A.4	Gràfic tridimensional d'eficiència, <i>clone rate</i> i <i>ghost rate</i> en funció de la tolerància i el nombre de <i>hits</i> amb cerques de tres veïns, sobre l'eix de tolerància.	73

A.5 Gràfic tridimensional d'eficiència, <i>clone rate</i> i <i>ghost rate</i> en funció de la tolerància i el nombre de <i>hits</i> amb cerques de quatre veïns, sobre l'eix de tolerància.	73
A.6 Gràfic tridimensional d'eficiència i temps d'execució en funció de la tolerància i el nombre de <i>hits</i> per esdeveniment, amb cerques de tres veïns.	74
A.7 Gràfic tridimensional d'eficiència i temps d'execució en funció de la tolerància i el nombre de <i>hits</i> per esdeveniment, amb cerques de quatre veïns.	74
A.8 Gràfic tridimensional de la suma dels valors normalitzats d'eficiència, <i>clone rate</i> , <i>ghost rate</i> i temps d'execució, referents als tres paràmetres de veïns diferents a estudiar, en funció de la tolerància i el nombre de <i>hits</i> per esdeveniment.	75
A.9 Diagrama de caixes de la variació de <i>clone tracks</i> en funció del nombre de <i>hits</i> per esdeveniment de l'algoritme original i l'optimitzat. La mitjana de la distribució es marca amb un triangle verd i la mediana amb una línia groga.	75
A.10 Diagrama de caixes de la variació de <i>ghost tracks</i> en funció del nombre de <i>hits</i> per esdeveniment de l'algoritme original i l'optimitzat. La mitjana de la distribució es marca amb un triangle verd i la mediana amb una línia groga.	76

Índex de taules

2.1	Posicions sobre l'eix z dels mòduls del VELO.	11
5.1	Resultats generals de la reconstrucció de l'esdeveniment 0.	40
5.2	Resultats de la reconstrucció de l'esdeveniment 0, desglossat segons el tipus de traces trobades.	40
5.3	Especificacions del <i>hardware</i> utilitzat per executar els algoritmes.	43
6.1	Resultats generals de la reconstrucció de l'esdeveniment 0 amb l'algoritme optimitzat.	54
6.2	Resultats de la reconstrucció de l'esdeveniment 0 amb l'algoritme optimitzat, desglossat segons el tipus de traces trobades.	55

Agraïments

Voldria expressar el meu reconeixement i agraïment a tots els col·laboradors del departament que, amb el seu suport i ajuda, han contribuït la realització d'aquest treball de final de grau.

En especial al Doctor i tutor Xavier Vilasís Cardona, per la seva confiança, consells, bon humor i dedicació durant tot el desenvolupament de projecte.

Gràcies a la família, als amics i a l'Àlex, per la paciència i companyia en moments de treball intens, nervis i també felicitat.

Capítol 1

Introducció

1.1 L'LHC al CERN

L'any 1954 es va fundar l'Organització Europea per la Investigació Nuclear a Suïssa, també anomenada CERN, de l'acrònim del francès *Conseil Européen pour la Recherche Nucléaire*, amb l'objectiu d'establir un centre de recerca de física fonamental de primer ordre a Europa. Durant els seus inicis, el focus principal de recerca es basava en l'estudi del nucli dels àtoms. Actualment, havent aprofundit molt més en el coneixement de la matèria, la seva principal àrea de recerca ha passat a ser la física de partícules, en específic, l'estudi dels constituents fonamentals de la matèria i les forces que actuen entre ells. D'altra banda, es beneficia de la col·laboració de més de 10.000 científics de prop de 60 països diferents.

El principi que regeix tots els processos d'experimentació realitzats al CERN és la colisió de partícules a una velocitat pròxima a la de la llum. D'aquesta manera, és possible observar el comportament de les partícules i la seva interacció en condicions similars a les del *Big Bang*, amb l'objectiu d'extreure tota la informació possible sobre les lleis fonamentals de la natura. I és que la particularitat més interessant de les seves instal·lacions és que posseeixen l'accelerador de partícules més gran del món, l'LHC [13] de l'anglès *Large Hadron Collider*.

Concretament, LHC és un accelerador circular de 27 km de diàmetre, situat a 100 metres sota terra, que esdevé l'últim element d'una cadena d'acceleradors de núvols de protons, també anomenats *beams*, tal com es pot veure a l'esquema de la instal·lació de la figura 1.1. El primer dels elements d'aquesta cadena és l'Accelerador Lineal 2 (*Linac 2*), que rep els protons des d'una bombona d'hidrogen i inicia el procés d'acceleració. A continuació, els *beams* són injectats al següent accelerador, *Proton Synchrotron Booster* (PSB), seguidament es traspassen al *Proton Synchrotron* (PS) i posteriorment al *Super Proton Synchrotron* (SPS), el tercer i el quart acceleradors respectivament. Finalment es fa una última transferència cap a les vies de l'LHC, on els *beams* es reparteixen per vies en sentits contraris. Dins d'aquest últim accelerador les partícules s'acceleren fins a la velocitat de $0.999999991c$ mitjançant imants superconductors operant a la temperatura de -271°C , més baixa que a l'espai

exterior, i arriben a aconseguir l'energia de 7 TeV.

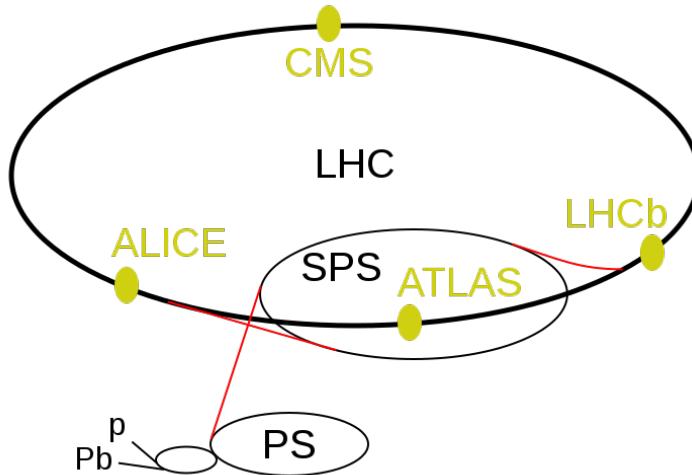


Figura 1.1: Esquema de la cadena d'acceleradors del CERN

Els *beams* circulen durant hores dins de l'LHC en condicions normals de treball, i els dos feixos de partícules de sentit contrari, es fan col·lidir en quatre punts estratègics de l'anell. Aquests corresponen als quatre detectors principals d'LHC, que són ATLAS, CMS, ALICE i LHCb [8].

ATLAS [2] (*A Toroidal LHC Apparatus*) és un detector de propòsit general que experimenta sobre temes molt diversos, des del bosó de Higgs¹ fins a partícules que podrien formar matèria fosca. Una particularitat important d'aquest detector és que les col·lisions es produïxen al centre d'aquest, permetent estudiar el comportament de les partícules posteriors a l'impacte en totes direccions.

Amb el mateix propòsit general, el detector CMS [1] (*Compact Muon Solenoid*) presenta una altra manera d'estudiar la física que envolta el Model Estàndard amb una tecnologia diferent. Està construït al voltant d'un immens imant cilíndric que genera un camp magnètic de 4 tesla (equivalent a 100.000 vegades el camp magnètic de la Terra). És un dels experiments amb major col·laboració científica internacional de la història.

ALICE [3] (*A Large Ion Collider Experiment*) és un detector d'ions de propòsit específic, optimitzat per estudiar les col·lisions de nuclis de plom, que són aproximadament 200 vegades més pesats que els protons.

Finalment, el detector LHCb [5] (*Large Hadron Collider beauty*) s'especialitza en investigació de les diferències entre la matèria i l'antimatèria observant una partícula anomenada *beauty quark* o quark b. A diferència d'altres detectors, aquest no envolta tot l'espai de la

¹El 4 de juliol de 2012 els experiments ATLAS i CMS van anunciar l'observació d'una partícula consistent amb la predicció del bosó de Higgs [27].

collisió sinó que mitjançant diversos sub-detectors es centra únicament en aquelles partícules en una sola direcció, cap endavant des de la col·lisió.

A banda dels quatre detectors més grans mencionats anteriorment, també es duen a terme altres experiments com TOTEM, LHCf i MoEDAL, entre d'altres, que aprofiten les instal·lacions facilitades pels grans experiments per a fer estudis de magnitud inferior.

1.2 Cronologia dels Experiments

Des de la primera posta en funcionament complet d'LHC el 10 de setembre de 2008, moment en el qual van començar a circular protons per primera vegada dins de l'accelerador, els processos d'experimentació i presa de dades no han sigut continus sinó que s'han dividit en períodes diferenciats i numerats, anomenats *Runs*. La durada aproximada d'aquests períodes d'experimentació ha sigut, fins al moment, de tres anys, en els quals es duen a terme les collisions dins de cadascun dels detectors i s'emmagatzemen les dades dels respectius sensors a una granja de servidors pel seu posterior anàlisi.

Un cop recollides totes les dades possibles, en acabar el període de *Run*, es procedeix a un període de pausa, anomenat *Long Shutdown*, en el que es procedeix a fer el manteniment de tots els sistemes de *hardware* que ho requereixin així com les possibles millors dels detectors dissenyades amb anterioritat, tant de *hardware* com de *software*. Degut a les condicions extremes en les quals es troben els sensors, tant de temperatura com de radiació, després d'un període de *Run* és necessari substituir gran part dels equips que queden malmesos.

Fins al moment, s'han dut a terme a LHC dos períodes de *Run*. El primer, *Run 1*, va iniciar-se l'any 2010 i va acabar el dia 16 de febrer de 2013, moment en el qual va començar l'*LS1* (*Long Shutdown 1*) i passats dos anys de pausa es va reprendre el fil d'experimentacions el 2015 amb l'inici del *Run 2*, període en el qual ens trobem actualment.

Tal com es pot veure a la cronologia de la figura 1.2, s'espera l'inici de l'*LS2* l'1 de juliol de 2018, que durarà fins al gener de 2020. Aquest serà el moment en el qual es duran a terme tots els canvis que actualment estan en fase de disseny. Es tracta doncs, d'un període clau de recerca per trobar, dissenyar i simular les actualitzacions més òptimes que seran aplicades als detectors de l'LHC per millorar el rendiment i funcionament dels sistemes pel futur *Run 3*, que començarà l'any 2020.

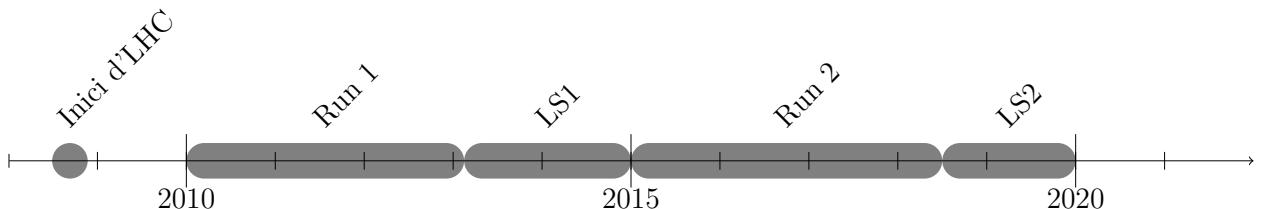


Figura 1.2: Cronologia de l'experimentació d'LHC

1.3 L'experiment LHCb

Tal com s'ha esmentat en el punt anterior, LHCb és un dels quatre detectors principals del LHC. La característica que el diferencia és que només s'ha instrumentat la regió cap endavant (*forward*) respecte al punt de col·lisió, amb l'objectiu d'aprofitar el fet que els hadrons b (partícules que contenen els quarks b, objectius de l'estudi) decauen en una immensa majoria dins d'un rang estrictament acotat. Aquesta característica permet focalitzar els sensors del detector en un con de dimensions relativament petites i conseqüentment poder aconseguir una millor resolució i sensibilitat dels fenòmens que hi succeeixen.

D'una forma més tècnica, el detector es pot descriure com un espectròmetre d'un sol braç amb una cobertura d'angle polar de 10 a 300 mili-radians en l'eix horitzontal i de 250 mili-radians en l'eix vertical [19]. Està format per diferents sensors, cadascun amb un objectiu específic, i que, en conjunt, recullen informació sobre la trajectòria, moment, velocitat i altres característiques de les partícules que creuen el detector, per poder identificar-les posteriorment i analitzar el seu comportament.

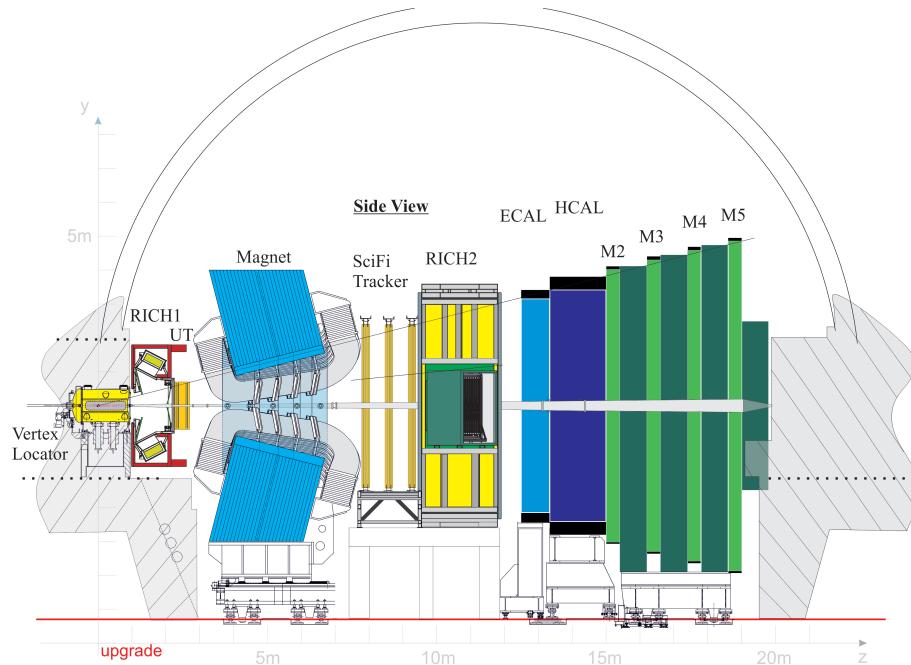


Figura 1.3: Vista lateral del detector LHCb.

A la figura 1.3 es pot veure una vista lateral del disseny de la nova versió actualitzada del detector, on únicament alguns dels mòduls anteriors han patit modificacions importants a banda dels respectius manteniments. La seva instal·lació està prevista pel pròxim LS2.

Començant a analitzar els sub-detectors d'LHCb d'esquerra a dreta (figura 1.3) es troba, en primer lloc, el ***Vertex Locator*** (VELO). És el detector més pròxim a la regió d'interacció

de les partícules i el seu objectiu principal és determinar el punt exacte on s'ha produït el vèrtex de la col·lisió. Gràcies al fet que el VELO envolta tota l'àrea on es poden produir els xocs de partícules, permet extreure informació sobre les traces que cauen fora l'acceptació de la resta de detectors d'LHCb, entre elles, les que es mouen en sentit contrari a la geometria del detector (*backward*). Les característiques del VELO s'expliquen més en detall al capítol 2.

Seguidament es troba el primer dels dos detectors ***Ring Imaging Cherenkov*** (RICH). Ambdós són encarregats de mesurar les emissions de l'efecte Cherenkov. Aquest fenomen succeeix quan una partícula viatja per un medi més ràpid del que ho fa la llum (en el mateix medi) i, d'una manera comparable a l'explosió sonora provocada per un avió en creuar la barrera del so, es produeix un con de llum que el RICH projecta sobre un vector de sensors mitjançant una estructura de miralls. El con de llum generat en creuar el medi, gasós en el cas del RICH, presenta un radi proporcional a la velocitat de la partícula, informació utilitzada per la seva posterior identificació.

Just abans de l'imant es troba l'***Upstream Tracker*** (UT). Es compon de quatre capes planes de detecció amb sensors de silici incorporats i, de la mateixa manera que les estacions del *SciFi Tracker*, les dues capes intermèdies del detector presenten una petita inclinació respecte l'eix *y*. Aquest detector extreu informació sobre la trajectòria de les traces de les partícules que el creuen.

A continuació es troba l'**Imant**. És un element clau per la identificació de les partícules, que aporta informació sobre el seu moment mesurant el radi de curvatura que pateix la trajectòria de cada partícula en presència del camp magnètic generat. L'Imant consta de dues bobines, cadascuna amb un pes de 27 tones, muntades sobre un marc d'acer i generen un camp integrat en longitud de 4 Tm que provoca que la trajectòria, inicialment recta, de les partícules carregades que creuin l'imant, es corbin en direccions oposades en funció del signe de la seva càrrega. A causa, precisament, d'aquesta modificació de les traces, l'imant marca una divisió en el detector que el separa en dues parts: l'anterior a l'imant és anomenada *Upstream* i la posterior *Downstream*.

Entrant als sensors de la part *Downstream* es troba en primer lloc l'anomenat ***Scintillating Fibre Tracker*** o *SciFi Tracker* és un detector format per tres estacions (T1, T2 i T3) que detecten la posició de les partícules que creuen els seus sensors i en reconstrueixen la trajectòria. Cadascuna de les tres estacions està composta per quatre capes de detecció en les que, de la mateixa manera que el detector UT, les capes intermèdies presenten una petita inclinació.

Seguidament es troba el segon detector RICH, el **RICH2**, que té el mateix propòsit i funcionament que el primer.

El següent element del detector LHCb és el calorímetre. Està dissenyat amb l'objectiu de frenar les partícules a mesura que passen pel detector i mesurar així la quantitat d'energia que perd cadascuna en disminuir la velocitat. Concretament LHCb presenta dos calorímetres diferenciats i amb funcions específiques. El primer, anomenat ***Electromagnetic Calorimeter*** (ECAL) és el responsable de mesurar l'energia de les partícules lleugeres, com els electrons i els fotons. El segon, anomenat ***Hadron Calorimeter*** (HCAL) és un calorímetre

d'hadrons, encarregat de mesurar l'energia de protons, neutrons i altres partícules formades per quarks. Ambdós detectors tenen una estructura de capes formada per diversos parells de panells, de metall i de plàstic. El principi de funcionament es basa en mesurar la quantitat de llum ultravioleta que es genera quan les partícules secundàries, derivades del xoc d'una partícula pesada contra la capa de metall, interaccionen amb la capa de plàstic. Els calorímetres són essencials per identificar les partícules que no tenen càrrega elèctrica, com són els fotons i els neutrons.

Finalment, es troben les **Estacions de Muons** o *Muon Stations* (M2-M5 a la figura 1.3). Els muons, unes partícules molt semblants als electrons, sovint són presents després del ràpid decaïment dels mesons B², per tant, detectar aquestes partícules permet reconstruir el pas dels mesons B que s'han descompost dins del detector. El sistema està format per cinc estacions amb una àrea de cobertura incremental on cadascuna conté petites cambres que interaccionen amb els muons i, mitjançant èlectrodes, es detecta de forma binària per quines cambres hi ha passat un muó.

El conjunt de dades extretes de cadascun dels nou sub-detectors s'utilitza per reconstruir l'esdeveniment d'una col·lisió de partícules detalladament amb l'objectiu d'analitzar els fenòmens que hi tenen lloc.

1.4 Condicions de Funcionament

Després del primer període d'experimentació d'LHC, en el que no es va observar cap fenomen més enllà del Model Estàndard³, es va conoure que era necessària una major precisió del detector per observar nous fenòmens. A tal efecte, durant l'LS1 es van dur a terme una millora significativa amb l'increment de la velocitat nominal d'experimentació de 20 MHz a 40 MHz. Aquest fet va suposar una reducció del temps transcorregut entre dues collisions del 50%, passant de 50 ns a 25 ns, i un augment de la lluminositat instantània⁴ d' $1.5 \times 10^{32} \text{ cm}^{-2}\text{s}^{-1}$ [18] a $1 \times 10^{34} \text{ cm}^{-2}\text{s}^{-1}$ [4].

Amb aquestes condicions, l'LHCb es va veure obligat a reduir la freqüència d'adquisició de dades del disseny actual pel *Run 2* a causa de la limitada tecnologia del moment, limitant el nombre d'esdeveniments emmagatzemats mitjançant un filtre de maquinari, també anomenat *trigger*. Aquest filtre està dividit en dues parts, la primera, formada per un *trigger* de maquinari (*hardware trigger*) que redueix la freqüència de collisions a tenir en compte a 1 MHz sense cap tipus d'anàlisi previ. La segona part consta de dos nivells de *trigger* de programari (*software trigger*), anomenats HLT1 i HLT2 (*High Level Trigger* 1 i 2)[12].

²Els mesons B són partícules elementals compostes per la combinació d'un anti-quark *bottom* i un quark *up*, *down*, *strange* o *charm*.

³Desenvolupat a principis dels 70, és una teoria que explica com interactuen els blocs més bàsics de la matèria, regits per les quatre forces fonamentals i que ha predit gairebé tots els resultats experimentals observats fins al moment.

⁴Mesura del nombre de collisions que es poden produir per cm^2 i per segon.

Primerament, l'HLT1 s'encarrega de fer una ràpida reconstrucció de la col·lisió, també anomenada esdeveniment, que s'utilitza per decidir si els fenòmens que hi han tingut lloc són prou rellevants com per al seu posterior anàlisi en més deteniment. Aquest procés es realitza amb un temps de 35ms per esdeveniment i filtra la sortida de dades fins a reduir-la a una velocitat de 150 kHz. La segona part del *software trigger*, l'HLT2 és l'encarregat d'enviar les dades corresponents als esdeveniments a emmagatzemar pel seu posterior anàlisi a una velocitat de 12.5 kHz.

Si es compara el funcionament de tot el sistema de *trigger* del *Run 1* i del *Run 2*, com es pot veure a la figura 1.4, s'ha aconseguit una gran millora en el temps de reconstrucció dels esdeveniments i en el procés de decisió d'emmagatzematge observant la diferència de velocitats de sortida les dades del *trigger*. Tot i això, el sistema segueix sense ser capaç de processar tots els esdeveniments produïts a LHC.

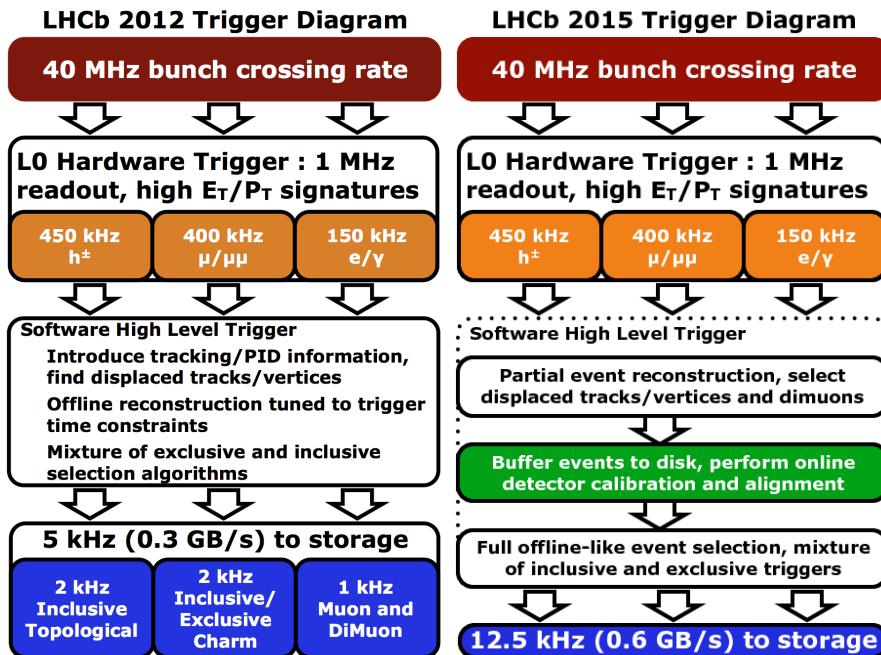


Figura 1.4: Diagrames del *trigger*, a l'esquerra corresponent al *Run 1* i a la dreta corresponent al *Run 2*.

Així doncs, el disseny del detector proposat per al futur *Run 3* té com a objectiu principal augmentar la capacitat de lectura de dades sense descartar cap esdeveniment des d'un primer nivell de *hardware*, és a dir, tractar directament mitjançant *triggers* de *software* els 40 MHz de dades generades per l'LHC [10]. Cal tenir en compte que s'ha determinat, mitjançant estudis, que un de cada quatre esdeveniments produïts a LHC no conté cap col·lisió de partícules, així que una quarta part de les dades es descarta i la freqüència de processat queda reduïda a 30 MHz, com es pot veure al diagrama de la figura 1.5.

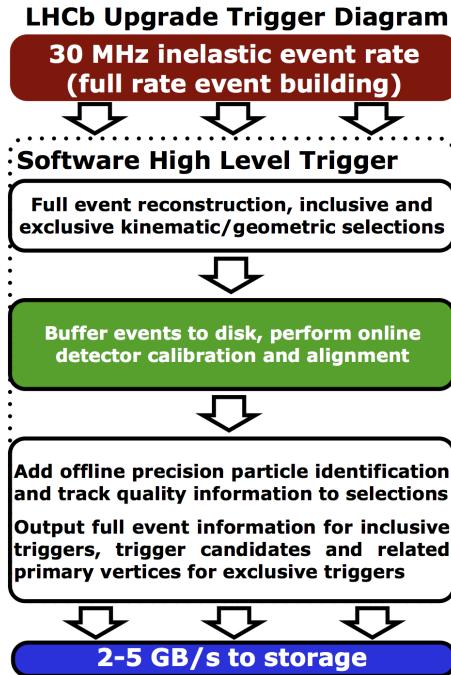


Figura 1.5: Diagrama del *trigger* corresponent al *Run 3*.

El principal obstacle que suposa aquesta futura millora del sistema de *trigger* és el temps de processament dels esdeveniments d'LHCb, ja que la capacitat computacional actual no permet emmagatzemar prou ràpidament totes les dades generades pel detector i, per tant, s'han de discriminar els esdeveniments que no són prou rellevants per al seu posterior ànalisi d'aquells que ho són. El repte al que s'enfronta doncs LHCb és dissenyar i implementar un software capaç de reconstruir un esdeveniment a partir de les dades de sortida del detector que contingui informació suficient per decidir si és prou rellevant per a emmagatzemar-lo i estudiar-lo detingudament o no, i tot això en un temps inferior a 25 ns.

Capítol 2

Detector VELO

2.1 Descripció General

El *VERtex LOcator* (VELO) és el detector de vèrtex de silici que envolta la regió d'interacció a LHCb. La seva principal tasca és reconstruir els vèrtexs de les desintegracions de partícules així com proporcionar informació per la reconstrucció de les traces necessària per al sistema de *trigger*. Gràcies al fet que és l'únic detector de l'experiment amb una cobertura completa de la regió de col·lisió de les partícules, aporta informació molt rellevant sobre les partícules que cauen fora l'abast de la resta dels sub-detectors. Concretament en direcció *backward*. No només aporta informació sobre el vèrtex primari de la col·lisió, sinó també de les partícules més pesades i més inestables que, al tenir una vida més curta, avancen uns pocs centímetres dins del mateix detector abans de tornar a desintegrar-se. Els punts on es produeix aquest fenomen s'anomenen vèrtexs secundaris.

Com ja s'ha mencionat a l'apartat anterior, el sistema de *trigger* actualment instal·lat serà actualitzat per passar a treballar directament a una velocitat de lectura de dades de 40 MHz en comptes d'1 MHz. Per fer possible aquesta restricció és necessària una actualització de tota l'electrònica dels sub-detectors que permeti complir els requeriments específics, cada vegada més complexes. Amb la millora del VELO que es durà a terme a l'LS2 s'espera aconseguir un bon rendiment a una lluminositat de $2 \times 10^{33} \text{ cm}^{-2} \text{s}^{-1}$ i acumular 50 fb^{-1} ¹ de dades.

2.2 Disseny de l'Actualització

L'única manera de complir amb les especificacions anteriors és substituint els sensors de silici i l'electrònica del detector per un nou disseny basat en sensors de píxels híbrids [11]. Aquest és l'anomenat **VeloPix**. Com es pot veure a la figura 2.1, està format per dues meitats retràctils, cadascuna de les quals conté un total de 26 mòduls de detecció en forma d'L.

¹La integral de la lluminositat en el temps s'anomena lluminositat integrada. És una mesura de la quantitat de dades obtingudes i es mesura en unitats inverses de secció eficaç: $1/\text{fb}$ o fb^{-1} - femtobarn^{-1}

Cada mòdul està format per 12 matrius de 256×256 píxels híbrids de silici de $55 \times 55\mu\text{m}^2$ cadascun. El sistema presenta una lectura binària de cada píxel on cadascun dels *hits*² queda marcat en temps, etiquetat i enviat immediatament per al seu processament.

La distribució de cadascun dels mòduls al llarg de l'eix z del detector s'ha modificat respecte al disseny anterior amb l'objectiu de mantenir l'acceptació tot i haver reduït la distància entre la línia del feix de partícules i el primer punt de mesura. Es defineix doncs el terme "estació" com al conjunt de dos mòduls equivalents al mateix índex de la meitat dreta i l'esquerra, on cadascun dels mòduls consta del seu propi sistema de refrigeració i lectura de dades.

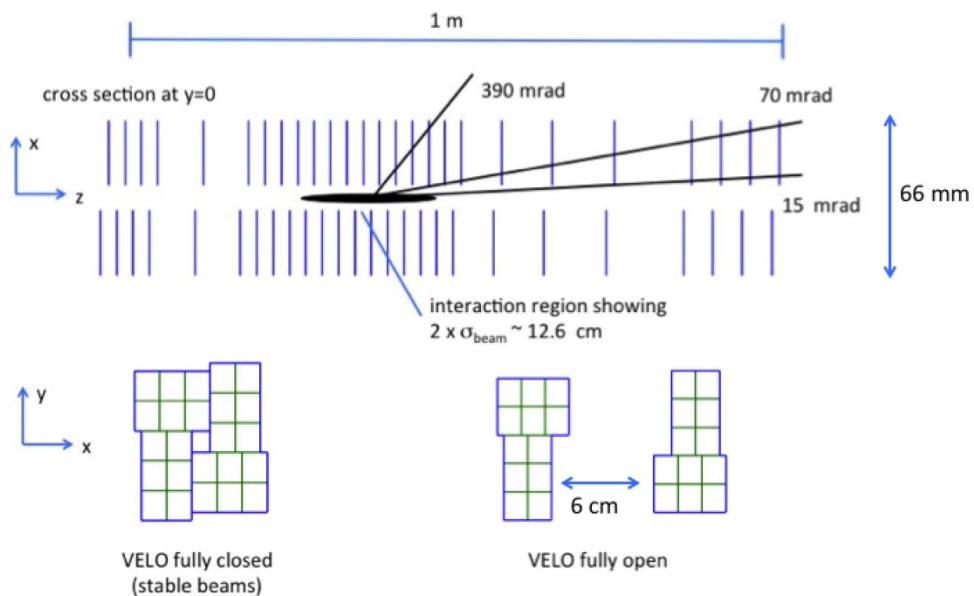


Figura 2.1: Disseny esquemàtic del VeloPix.

Per la reconstrucció i tractament de les dades del sensor, cal tenir present la numeració i posició dels mòduls dins el detector, llistats a la taula 2.1: el costat esquerre (costat A) està en direcció de les x positives, el costat dret (costat C) està en direcció de les x negatives. La posició sobre l'eix z dels mòduls es dóna en mm des de la zona d'interacció, equivalent a $z = 0$.

²Hit: marca que registra un sensor provocada pel pas d'una partícula.

Side	Module z-position [mm]								
	-277.0	-252.0	-227.0	-202.0	-132.0	-62.0	-37.0	-12.0	13.0
A	38.0	63.0	88.0	113.0	138.0	163.0	188.0	213.0	238.0
	263.0	325.0	402.0	497.0	616.0	661.0	706.0	751.0	
C	-289.0	-264.0	-239.0	-214.0	-144.0	-74.0	-49.0	-24.0	1.0
	26.0	51.0	76.0	101.0	126.0	151.0	176.0	201.0	226.0
	251.0	313.0	390.0	485.0	604.0	649.0	694.0	739.0	

Taula 2.1: Posicions sobre l'eix z dels mòduls del VELO.

A la figura 2.2 es pot veure el marge d'acceptació del detector dins del VELO. Aquelles traces que hi passin seran de gran qualitat, ja que deixaran marca també al llarg dels diferents sub-detectors d'LHCb, fet que permetrà tenir més precisió en la seva reconstrucció.

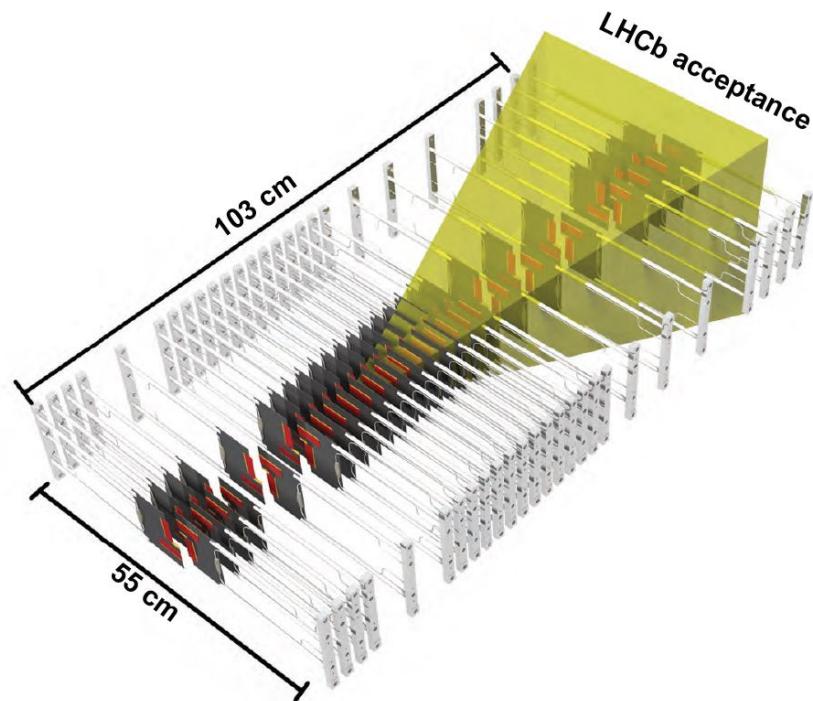


Figura 2.2: Representació gràfica de l'àrea d'acceptació dins del VeloPix.

2.3 Classificació de Traces

Per poder reconstruir correctament un esdeveniment pel sistema de *trigger* cal tenir present la informació dels diferents sub-detectors d'LHCb, que proporcionen informació sobre la trajectòria, moment i velocitat, entre d'altres, de les partícules resultants de la col·lisió de dos protons. Concretament, en la reconstrucció de les traces de les partícules hi intervenen tres sub-detectors que, a la vegada, determinen la classificació de les traces (figura 2.3). Aquests són el VELO, l'UT i el SciFi Tracker.

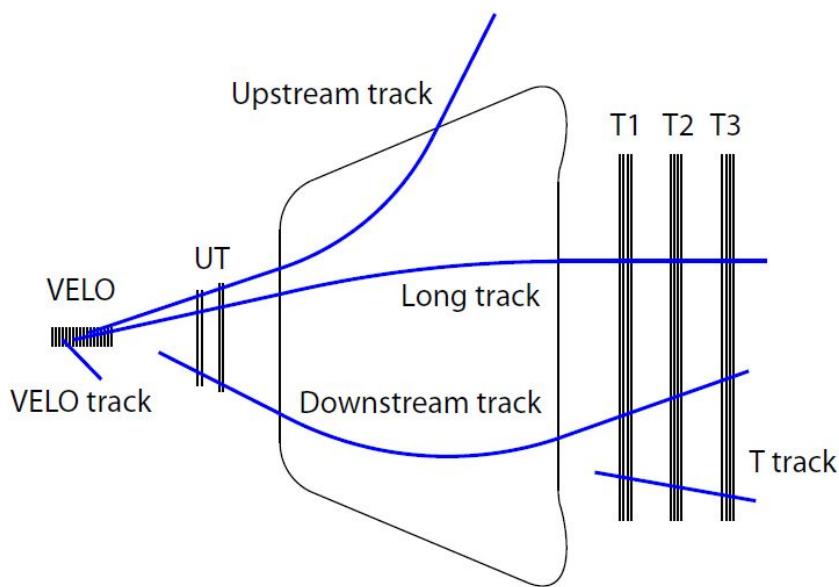


Figura 2.3: Representació del tipus de traces sobre el detector.

Les traces més significatives són les anomenades ***long tracks***, que comencen al VELO i passen pels dos sub-detectors restants. Al deixar rastre tant abans com després de l'Imant, aporten informació molt precisa sobre el moment de la partícula.

Les traces mesurades a l'UT i a les estacions del SciFi Tracker s'anomenen ***T tracks*** o ***downstream tracks*** i, tot i que no s'utilitzen per a l'anàlisi físic, són importants per la reconstrucció de les filles de les partícules que han viscut prou temps al detector per veure la seva desintegració allunyada del vèrtex de la col·lisió.

Les traces mesurades al VELO i a l'UT són anomenades ***upstream tracks*** i s'utilitzen per reconstruir les traces de partícules amb moment baix que no arriben a passar pel SciFi Tracker a causa de la desviació provocada per l'Imant.

Per últim, les ***VELO Tracks*** són les traces que únicament han deixat rastre dins del VELO. S'utilitzen principalment com a principis de traces existents a la resta del detector

i per trobar el vèrtex de la col·lisió. Concretament, una partícula és reconstruïda com una *VELO Track* si ha deixat rastre a tres o més mòduls del detector.

Els processos de reconstrucció de traces a partir de les dades dels detectors es duen a terme mitjançant algoritmes de programari. Una manera d'avaluar-ne la precisió és amb l'**eficiència** (ε), una figura de mèrit que utilitza els resultats de les simulacions Monte Carlo realitzades sobre el disseny del detector per comparar el nombre de traces correctament reconstruïdes envers el nombre total de traces que es poden reconstruir en un esdeveniment. Una traca serà considerada correctament reconstruïda si almenys concorda en un 70% amb trajectòria d'una partícula simulada.

$$\varepsilon = \frac{N(\text{reconstructed} \& \text{reconstructible} \& \text{!electrons})}{N(\text{reconstructed} \& \text{!electrons})} \quad (2.1)$$

Com es pot veure a l'equació 2.1, en el càlcul de l'eficiència no es tenen en compte les traces produïdes pels electrons a causa de l'efecte Bremsstrahlung³, ja que complica la seva reconstrucció.

Cal tenir en compte que els processos de reconstrucció poden generar traces falses que no corresponguin a la trajectòria de cap partícula reconstituïble. Són les anomenades ***ghost tracks*** i s'utilitzen per quantificar una figura de mèrit de l'eficiència de reconstrucció anomenada *ghost rate*, que mesura la fracció de *ghost tracks* relatives a totes les traces reconstruïdes d'un esdeveniment. També pot donar-se el cas que més d'una traça sigui associada a una mateixa partícula de la simulació Monte Carlo, a les hores, únicament una d'elles és considerada correctament reconstruïda mentre la resta es comptabilitza com a ***clone tracks***. El nombre de *clone tracks* relatives a totes les traces reconstruïdes s'anomena *clone rate*.

2.4 VELo Tracking

El procés de reconstrucció dissenyat per l'actualització del VELo [6] comença amb la cerca de parelles de *hits*, que no formen part de cap altre traça ja reconstruïda, situats en estacions veïnes i que siguin compatibles, és a dir, que la distància entre ells sigui menor a l'acotada a $|dx/dz| < 0.4$, $|dy/dz| < 0.4$. Si una parella de *hits* compleix aquestes condicions, passarà a anomenar-se *seed*.

Un cop identificada una *seed*, es calcula la recta que formen els seus dos punts alineats i s'extrapolà en direcció *upstream*, cap a la resta de mòduls del detector. Els següents *hits* que es trobin més pròxims a la recta, amb una tolerància determinada per l'angle màxim de dispersió, seran inclosos a la recta juntament amb la *seed* inicial. Únicament aquelles traces amb només tres *hits* requereixen que cap dels seus *hits* sigui utilitzat en la reconstrucció d'altres possibles traces del mateix esdeveniment.

Amb la informació de totes les traces reconstruïdes és possible determinar quin ha estat el vèrtex de la col·lisió, anomenat vèrtex primari[16]. Però pot donar-se el cas que no només hi

³Radiació electromagnètica resultant de la desviació de la trajectòria d'una partícula carregada en passar a prop d'una altra partícula carregada.

hagi un sol vèrtex primari en un esdeveniment, i amb aquesta premissa, s'efectuen simulacions d'esdeveniments amb diferents nombres de vèrtex primaris, des d'un fins a un màxim de quinze, amb l'objectiu de comprovar l'eficiència de l'algorisme de reconstrucció en tots els casos possibles.

En comparació amb l'antic algorisme de reconstrucció del VELO, l'actualització presenta una eficiència més elevada. Però presenta un petit error: pel propi disseny, l'algorisme no busca un segon *hit* al mateix mòdul, fet que ocasionalment podria donar-se.

Quant al temps d'execució, a l'actual algorisme utilitzat al detector VELO, el temps mitjà de reconstrucció de totes les traces és de $3ms$ per esdeveniment. En canvi, amb el programari dissenyat per l'actualització, basat en reconeixement de patrons, s'ha aconseguit amb simulacions un temps d' $1.6ms$ per esdeveniment⁴ [17]. Tot i la millora de temps, els algoritmes dissenyats no s'han optimitzat encara, així doncs, s'espera una millora futura en el temps d'execució. Tanmateix, el rendiment actual es troba dins dels requisits de l'actualització.

⁴Aquest temps d'execució s'ha obtingut en un processador Intel Xeon L5520.

Capítol 3

Descripció de les Dades Reals

Abans de pensar en els possibles algoritmes a aplicar per la reconstrucció de trances del detector VELO, cal conèixer en detall el format de dades de sortida del detector, per poder adaptar i optimitzar correctament les implementacions a les dades reals. En aquest capítol s'explicarà en detall el format de dades del que es disposa així com la seva validació i estructura interna.

3.1 Simulació Monte Carlo

La simulació Monte Carlo (MC) és un software que proporciona prediccions del comportament teòric de les collisions de les partícules dins dels detectors d'LHC. El seu propòsit és generar esdeveniments amb el màxim detall possible. Les dades generades en aquestes simulacions s'utilitzen per provar i verificar el comportament dels detectors enfront de collisions reals i ajudar als físics a comprendre el comportament de les partícules observat en els resultats reals. D'aquesta manera, la simulació MC pren un rol molt important en els experiments, ja que s'utilitza també com a entrada dels algoritmes de reconstrucció reals, amb l'avantatge que es coneix l'anomenada “veritat de MC”, és a dir, les partícules que han originat les degudes reconstruccions de forma verídica. Així doncs, la comparació entre els resultats obtinguts en la reconstrucció de la simulació, juntament amb la veritat de MC i la reconstrucció de les dades extretes dels sensors en una colisió real, permet una millor interpretació dels fenòmens succeïts.

El procés de generació d'un esdeveniment MC es divideix en dos softwares de simulació diferenciats[25]. El primer, anomenat *Gauss Simulation*, és l'encarregat de determinar quines partícules seran creades en cada colisió de partícules en funció d'unes probabilitats determinades pel Model Estàndard de física, mantenint una certa aleatorietat.

A causa de la gran complexitat que presenten els diferents detectors d'LHC, es realitzen moltes simulacions MC especialitzades en un domini concret, que unides, proporcionen una visió global d'un esdeveniment. D'aquesta manera, existeixen diferents nuclis de programari especialitzats en simulacions de partícules concretes, així com de propòsit general. Alguns

exemples són PYTHIA i SHERPA, entre d'altres.

La segona part de la simulació, anomenada *Boole Digitalization*, s'encarrega de convertir la informació de les partícules generades per la simulació Gaussiana, a les dades de sortida dels sensors del detector com a resposta a les partícules simulades. D'aquesta manera, la informació simulada després de la digitalització presenta el mateix format que les dades de sortida d'una col·lisió real i permet la comparació dels resultats de forma directa.

El nucli de programari de la digitalització de Boole s'anomena GEANT4[9] i està programat en C++. Aquest programari requereix una descripció molt detallada de la geometria del detector per poder donar uns resultats els més pròxims a la realitat possibles, tot i tenir en compte que els calibratges realitzats als detectors reals poden diferir de la simulació.

La figura 3.1 mostra un diagrama representatiu del procés de creació d'una simulació MC, on es diferencien les dues etapes: generació d'esdeveniments, amb el nucli específic per la simulació desitjada dins d'una interfície que permet canviar els paràmetres de la simulació, i des d'on s'estreuen les dades de les partícules generades, anomenades *High Energy Physics Monte Carlo* (HepMC). A continuació, aquestes entren a la simulació del detector, que extreu les dades pertinents a la sortida dels sensors i crea una estructura de dades anomenada *Event Model*.

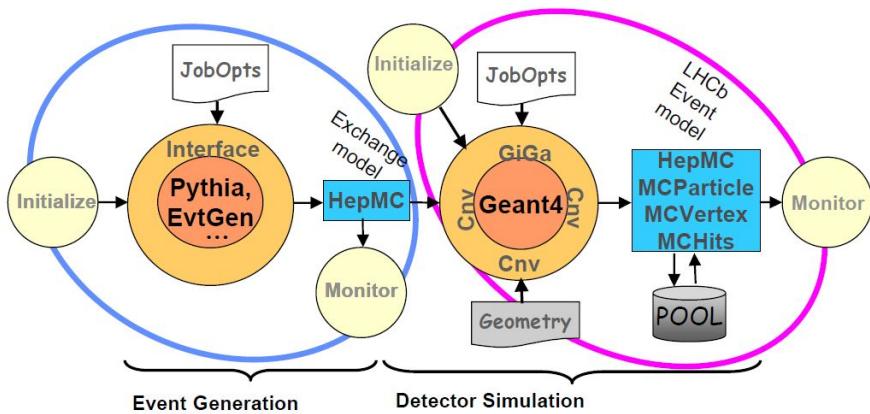


Figura 3.1: Diagrama del procés de generació d'una simulació Monte Carlo d'LHC.

3.2 *Event Model*

Amb l'objectiu de fer les dades més fàcilment accessibles a l'hora d'executar un algorisme, les dades de sortida de l'electrònica dels detectors corresponents a la reacció als passos de les partícules, són processades per crear una estructura de dades anomenada *Event Model*[26], que, en el cas del detector VELO, organitza la informació de cada *hit*, del qual únicament se'n saben les seves coordenades x , y i z , en funció dels mòduls del sensor on es troben, per facilitar el tractament posterior de les dades en els algoritmes de reconstrucció. L'estructura de l'*Event Model* consta de quatre classes d'objecte diferents que s'explicaran a continuació.

Un esdeveniment s'instancia com un objecte de la classe `event`, que consta de quatre atributs, tal i com es pot veure al diagrama de la figura 3.2.

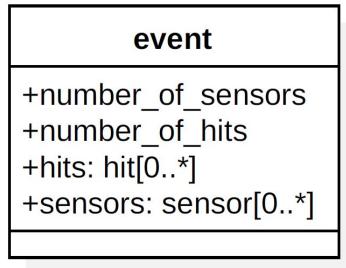


Figura 3.2: Diagrama de la classe `event`.

L'atribut `number_of_sensors` guarda el nombre de sensors, `number_of_hits` conté el nombre de *hits* totals que s'han detectat, `hits` és un *array* de la classe `hit` (figura 3.3) que emmagatzema les dades de cadascun dels *hits* de l'esdeveniment. Finalment, `sensors` és un *array* de la classe `sensor` (figura 3.3) que guarda la informació relativa a l'esdeveniment estructurada per cada sensor del detector.

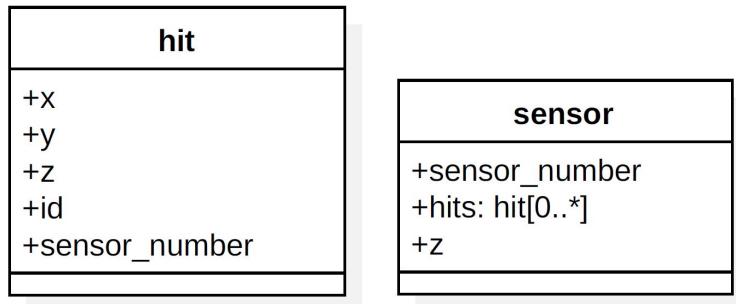


Figura 3.3: A l'esquerra, diagrama de la classe `hit`, a la dreta, diagrama de la classe `sensor`.

La classe `hit` s'estructura en cinc atributs: les tres coordenades de la seva posició `x`, `y` i `z`, un identificador únic, `id`. També guarda el número del sensor al qual pertany, `sensor_number`. La classe `sensor` també presenta un atribut amb l'índex del sensor `sensor_number`, un *array* dels *hits* que han passat per ell i la coordenada `z` de la posició del sensor.

Per altra banda, també es troba la classe `track` (figura 3.4), que servirà per agrupar tots aquells *hits* que hagin estat classificats en una mateixa traça.

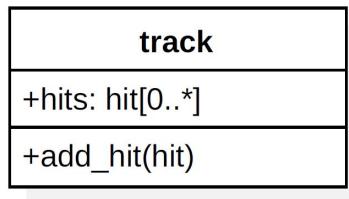


Figura 3.4: Diagrama de la classe `track`.

Així doncs, l'estructura de dades que es disposa, classifica els *hits* en funció del sensor (*z*) on han estat detectats, però aquesta informació de cadascun, és únicament relativa a la seva pròpia posició dins el detector, i per tant, independent de tots els altres *hits* detectats. Quan es procedeix a avaluar un *hit*, es disposa de la seva informació en el format següent[7]:

```
#hit_id {x_position, y_position, z_position}
```

Capítol 4

Mètode Clàssic: Transformada de Hough

Amb l'objectiu d'estudiar diferents mètodes de reconstrucció, es comença per estudiar un dels algorismes que han inspirat l'actual sistema de *forward tracking*, encarregat de relacionar les traces del VELO amb les traces reconstruïdes de la resta de detectors d'LHCb. Aquest algoritme és la transformada de Hough.

4.1 Principi de l'Algorisme

La transformada de Hough és una tècnica d'extracció de funcions utilitzada en anàlisi i processament digital d'imatges. El seu objectiu és detectar figures matemàtiques tals com rectes, circumferències o el·lipses mitjançant un procés de votació. Aquest procés de votació es realitza en un espai de paràmetres anomenat acumulador, d'on s'extreuen els candidats a objectes com els màxims locals. La versió clàssica de l'algorisme es basava en la identificació de línies rectes en una imatge, i posteriorment, es va expandir per identificar formes arbitràries, les més conegudes mencionades anteriorment. Actualment s'utilitza la versió de l'algorisme inventat per Richard Duda i Peter Hart l'any 1972, bategada amb el nom "Transformada de Hough Generalitzada".

Inicialment, l'algorisme va ser inventat per l'anàlisi de les fotografies d'una cambra de bombolles¹, l'any 1959, sota la patent de "Mètode i Mitjans pel Reconeixement de Patrons Complexos". A la figura 4.1 es mostra un exemple de les formes a reconstruir per l'algorisme de les imatges d'una cambra de bombolles.

¹Recipient omplert de líquid super escalfat transparent utilitzat per detectar partícules carregades elèctricament que el travessen.

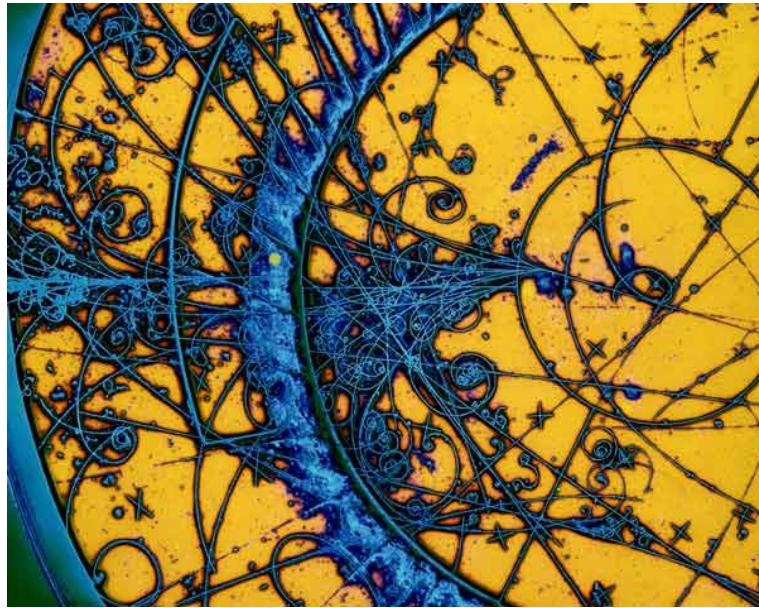


Figura 4.1: Imatge extreta del *Big European Bubble Chamber*, iniciat al CERN l'any 1973[21].

Respecte a l'aplicació més rellevant de l'algorisme per aquest treball, la detecció de línies rectes, cal saber que una recta es caracteritza amb dos paràmetres, seguint l'equació 4.1. On m correspon al pendent i n al coeficient de posició d'aquesta. D'aquesta manera es pot representar una recta com el punt (m, n) dins l'espai bidimensional de l'acumulador.

$$y = mx + n \quad (4.1)$$

Però en el cas de tenir una recta vertical, en procedir a fer el càlcul dels paràmetres, la pendent de la recta esdevé infinita i, per tant, seria necessari un espai acumulador infinit per poder comptabilitzar-la. Per aquest motiu, en comptes de l'equació clàssica, s'utilitza la forma normal de Hesse, definida per l'equació 4.2, que caracteritza una recta amb (ρ, θ) . On ρ és la distància normal de la recta a l'origen i θ és l'angle que formen l'eix x i la recta que uneix l'origen amb el punt més proper a la recta.

$$\rho = x \cos \theta + y \sin \theta \quad (4.2)$$

Donat un punt concret en un pla, totes les possibles línies que passin per aquell punt formaran una corba sinusoidal a l'espai (ρ, θ) , també anomenat espai de Hough, i que és única pel punt donat. Si es repeteix el procés per un segon punt alineat amb el primer, la corba generada a l'espai de Hough creuarà la primera corba en un punt concret: el punt corresponent als paràmetres de la recta que alinea els dos punts. El funcionament es repeteix amb cada punt alineat amb els anteriors, de manera que la recta que tingui més punts alineats d'un conjunt equivaldrà al punt màxim (el punt on es creuin més corbes) de l'espai de Hough.

4.2 Primera Implementació

Per fer una implementació de la transformada de Hough cal tenir present quina serà la dimensió de l'espai de Hough. Haurà de ser capaç de representar els punts de paràmetres de totes les possibles rectes que contingui la imatge a processar. Es dissenya doncs una estructura de tipus matriu amb els valors de ρ corresponents a la distància màxima normal al 0 (la diagonal de la imatge) i els valors de θ corresponents a tots els possibles angles de rectes dins la imatge (de 0° a 180°). A continuació es procedeix a iterar sobre cadascun dels punts (x, y) de la imatge, i es calculen els paràmetres (ρ, θ) de la recta que forma amb cadascun dels punts restants de la imatge, tal com exemplifica la figura 4.2. Finalment es comptabilitza aquella recta a l'espai de Hough incrementant en 1 la posició (ρ, θ) de la matriu acumuladora.

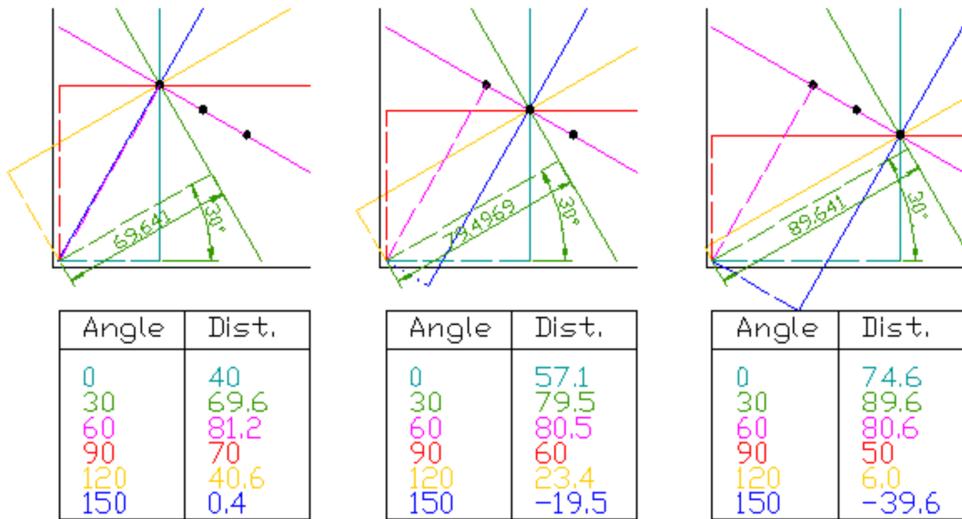


Figura 4.2: Procés d'extracció dels paràmetres de totes les possibles rectes amb diferents angles que passen pels diferents punts d'un conjunt en una imatge.

Amb l'objectiu de comprovar el funcionament de l'algorisme i entendre per complet la seva mecànica, s'ha implementat una versió de la transformada de Hough en llenguatge Python[14]. En primera instància s'ha dissenyat per tal de poder identificar rectes sobre una imatge bidimensional de dimensions 15×15 i amb un total de tres rectes, tal com es pot veure a la figura 4.3. A continuació s'ha creat l'espai de Hough, amb l'eix x acotat a $[0, 180]$ i l'eix y acotat a $[-22, 22]$, tenint present que 22 correspon a la diagonal màxima de la imatge ($\sqrt{WIDTH^2 + HEIGHT^2}$).

Per dur a terme el procés de votació de les rectes possibles s'ha implementat una iteració per cadascun dels punts de la imatge on es calculen els paràmetres corresponents a la recta que forma el punt i cadascun dels possibles angles θ (recordem, de 0 a 180). A la vegada,

els paràmetres s'indexen en un histograma² de dues dimensions, ponderant en cada iteració el punt de l'espai de Hough corresponent a la recta calculada.

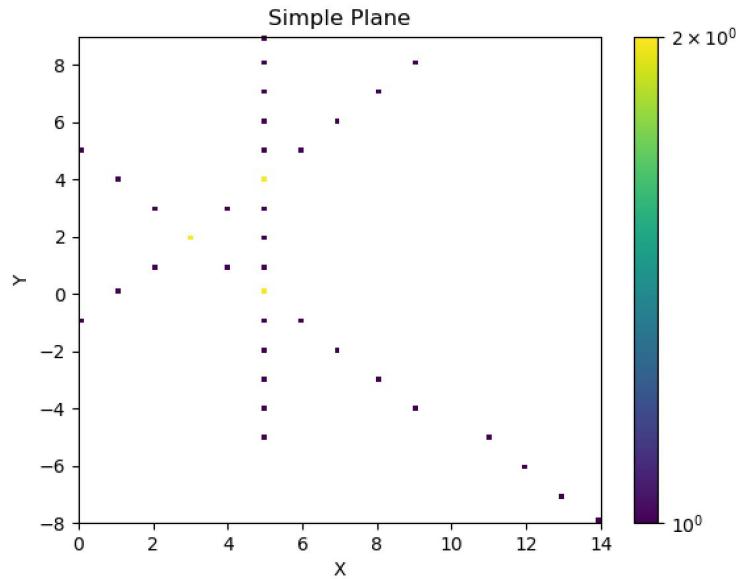


Figura 4.3: Conjunt de punts alineats en tres rectes sobre el pla x, y .

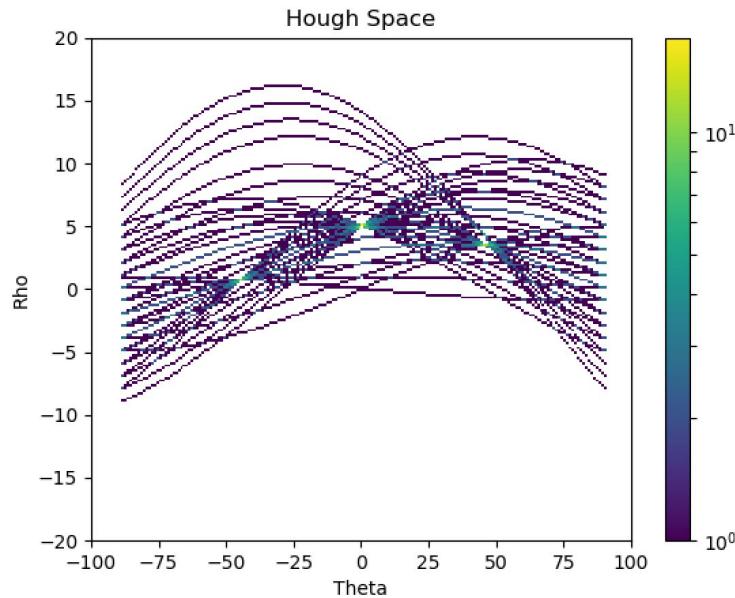


Figura 4.4: Espai de Hough resultant de la transformada sobre la imatge de la figura 4.3.

²Representació gràfica d'una variable en forma de barres on la superfície de cada barra és proporcional a la freqüència dels valors representats.

Com es pot observar a la figura 4.4, a l'espai de Hough resultant s'aprecien tres punts de convergència de les corbes, que també coincideixen amb els punts més votats. Per detectar-los a través de les dades de l'histograma, s'ha buscat el màxim valor (el més votat) d'entre totes les caselles i s'han extret els índexs dels eixos corresponents, d'aquesta manera es poden obtenir els valors ρ i θ de la recta representada pel punt màxim. S'ha repetit l'operació fins a trobar els tres punts més votats, corresponents a les següents coordenades:

$$\text{Recta 1: } (0, -45) \rightarrow 0 = x \cos(-45) + y \sin(-45)$$

$$\text{Recta 2: } (5, 0) \rightarrow 5 = x \cos(0) + y \sin(0)$$

$$\text{Recta 3: } (3.4, 45) \rightarrow 3.4 = x \cos(45) + y \sin(45)$$

4.3 Adaptació per un Sistema Tridimensional

Un cop comprovat el funcionament de l'algoritme per al reconeixement de restes en un pla, cal tenir en compte que les dades de sortida del detector formen un espai tridimensional format pels eixos x , y i z . Així doncs, per poder aplicar la transformada de Hough a la geometria del detector es treballa, en primera instància, amb projeccions de les dades sobre els plans XZ i YZ , per reduir-les a un sistema bidimensional.

Per poder provar la correcta detecció de rectes en un espai tridimensional a partir de projeccions s'ha creat una imatge tridimensional de dimensions $10 \times 10 \times 10$ amb dues línies diferents formades per punts, tal com es pot veure a la figura 4.5.

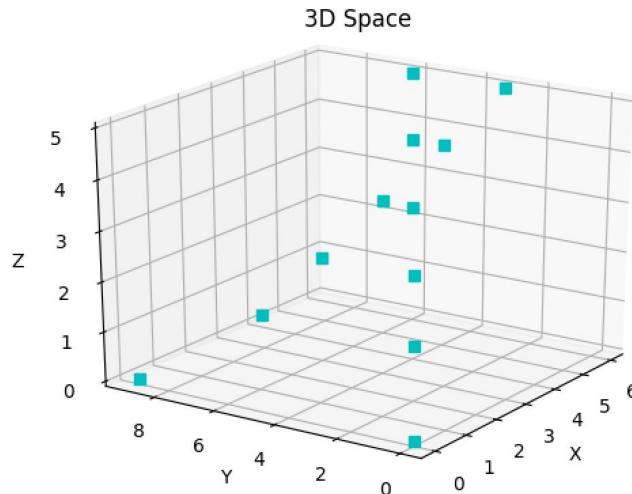


Figura 4.5: Conjunt de punts alineats en dues rectes tridimensionals.

A continuació, amb l'objectiu d'aplicar la transformada, s'han extret les dues projeccions abans explicades (figura 4.6). Sobre aquestes, s'ha aplicat l'algorisme implementat per

generar l'espai de Hough pertinent per cada projecció, tal com es pot veure a la figura 4.7, i poder detectar aquelles rectes més probables en cadascuna de les imatges.

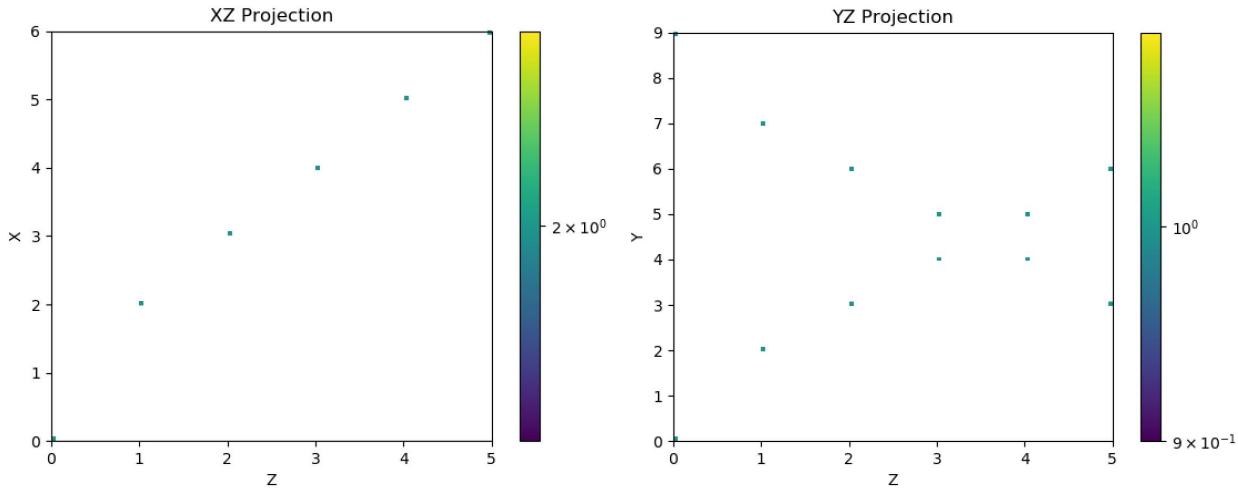


Figura 4.6: Projeccions de la imatge de la figura tridimensional 4.5 sobre el pla XZ a l'esquerra, i sobre el pla YZ a la dreta.

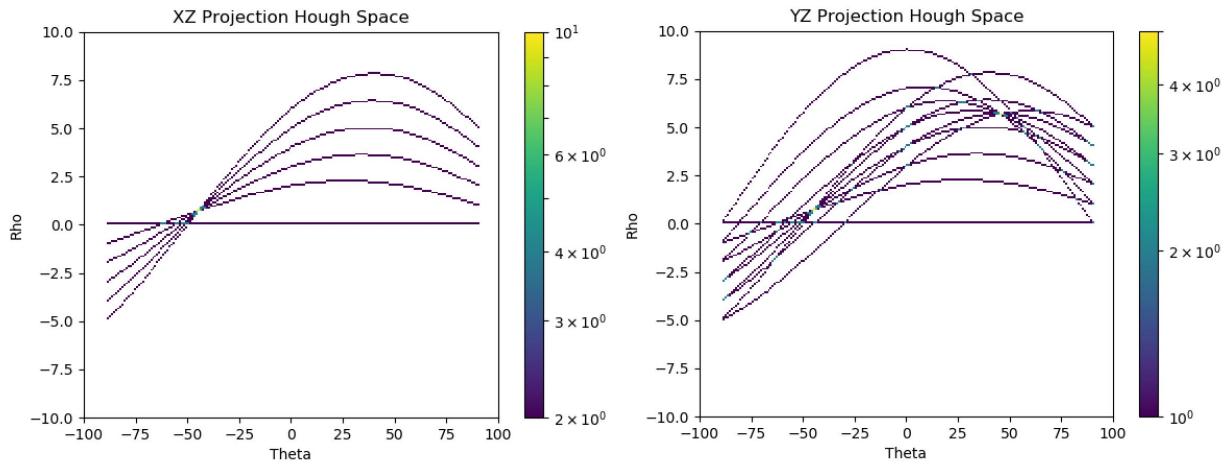


Figura 4.7: Conjunt de punts alineats en dues rectes tridimensionals.

Per poder saber quines rectes originals, en tres dimensions, hi ha a la imatge, cal relacionar el parell de rectes projectades que provenen d'una mateixa recta tridimensional. Per fer-ho, es parteix d'una premissa aplicable a les dades reals del sensor VELO: l'últim mòdul del sensor, aquell situat a la z més gran, serà el que tindrà els *hits* més espaiats entre si i, per tant, un *hit* únicament correspondrà a una traça. Amb aquesta suposició, es buscarà, per cada punt de l'últim mòdul de la imatge, quines són les rectes amb la distància normal

al punt més properes, en cadascuna de les projeccions, determinant les corresponents com a parella.

Finalment, amb les parelles de rectes de cada projecció, es pot definir una recta tridimensional a partir de les equacions de dos plans, que s'extrauran de la prolongació de cada projecció sobre l'eix normal a pla projectat. Així doncs les rectes originals de la imatge es definiran per les interseccions de les parelles de plans trobats.

Recta 1:

$$\text{Pla 1: } (A = 0, B = 0.812, C = -2, D = 0) \rightarrow 0.812y - 2z = 0$$

$$\text{Pla 2: } (A = -0.127, B = 0, C = 2, D = 0) \rightarrow -0.127x + 2z = 0$$

Recta 2:

$$\text{Pla 1: } (A = 0, B = 0.812, C = -2, D = 0) \rightarrow 0.812y - 2z = 0$$

$$\text{Pla 2: } (A = -17.457, B = 0, C = 2, D = 0) \rightarrow -17.457x + 2z = 0$$

4.4 Adaptació a les Dades Reals

Un cop comprovat el funcionament de l'algorisme a petita escala, s'ha adaptat per treballar amb les dades de sortida del sub-detector VELO, que contenen un nombre de *hits* variable entre 1000 i 5000, repartits entre els cinquanta-dos mòduls del detector.

Durant el procés d'implementació ha sorgit el primer punt en contra d'aquest algorisme: les dimensions de l'acumulador. En el cas dels exemples previs, el nombre de cel·les de l'espai de Hough tenia unes dimensions computables, però en el cas de les dades del sensor, les dimensions de l'acumulador esdevenen grans: es requereix de prop de 300.000 cel·les per obtenir la resolució equivalent al detector. Així doncs, el fet de construir una estructura de dades matricial d'aquesta mida resulta no computable a causa dels enormes requeriments de memòria que es necessiten. Per evitar-ho, s'han fet servir histogrames en comptes de matrius, com ja s'ha comentat anteriorment, ja que són estructures molt més optimitzades per que fa a memòria i permeten generar estructures amb la granularitat necessària per indexar totes les possibles rectes existents al detector.

En primer lloc, a les gràfiques de la figura 4.8 es poden veure les dades del sensor projectades sobre els eixos XZ i YZ , i des d'on es pot observar l'origen de la col·lisió així com la direcció de les trajectòries i l'acceptació del detector.

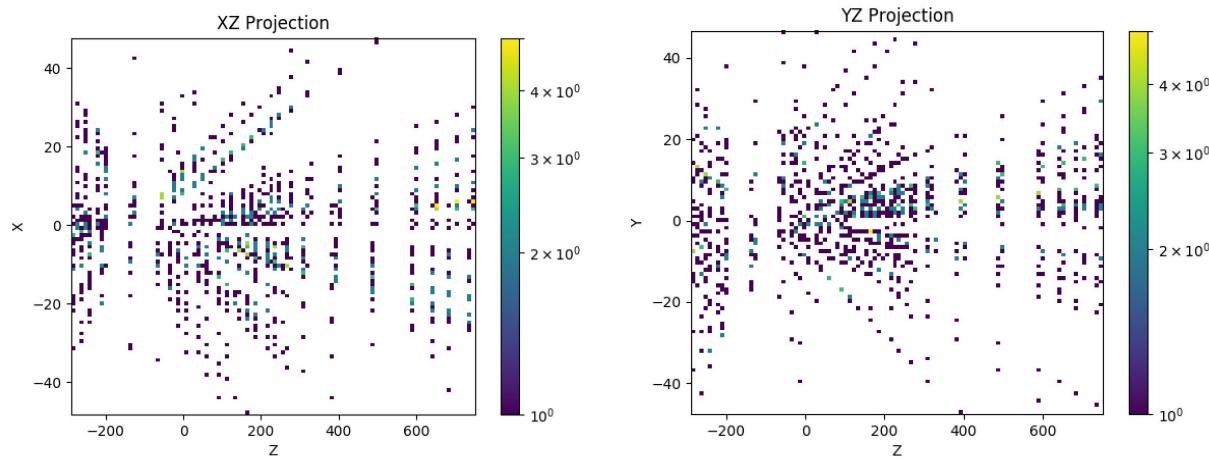


Figura 4.8: A l'esquerra, projecció sobre els eixos XZ d'un esdeveniment d'LHCb al sub-detector VELO. A la dreta, projecció sobre els eixos YZ .

Sobre les gràfiques anteriors s'aplica la transformada. Els resultats obtinguts de l'espai de Hough sobre la projecció XZ són els observats a la figura 4.9, i es pot veure com es genera una simetria respecte l'angle $\theta = 0^\circ$ i els punts més votats es concentren als punts pròxims a l'angle $\theta = 90^\circ$, corresponent a l'angle de la recta normal a la detectada, esdevenint doncs, les rectes més probables detectades únicament les que creuen l'origen de la projecció. Certament, les traces horitzontals i pròximes al punt $(0, 0)$ presenten un major nombre de punts alineats, tot i que no necessàriament han de pertànyer a la mateixa traça.

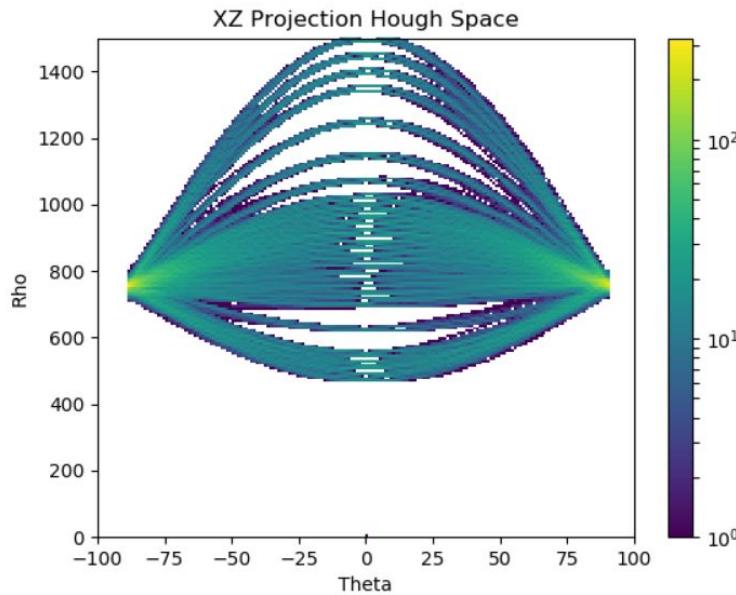


Figura 4.9: Espai de Hough resultant d'aplicar la transformada sobre la projecció XZ .

Així doncs, ja que no només s'han de detectar les traces horitzontals sinó també totes aquelles que cauen dins l'acceptació del detector, amb els resultats a partir de l'espai de Hough generat, és impossible detectar les traces com a màxims clars dins l'acumulador a causa de la gran quantitat de soroll generat per la transformada.

Si s'analitza també l'espai de Hough generat a partir de la projecció sobre els eixos XY (figura 4.10), s'observen resultats similars.

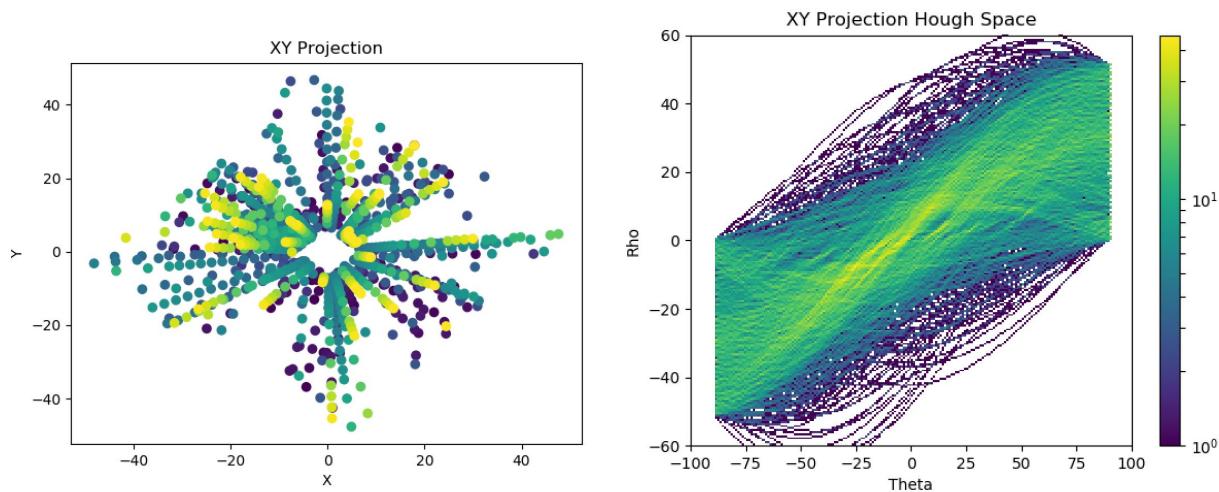


Figura 4.10: A l'esquerra, projecció dels *hits* d'un esdeveniment sobre els eixos XY , a la dreta, espai de Hough generat aplicant la transformada sobre la imatge de l'esquerra.

Tant la figura 4.9 com la 4.10, on es pot veure l'espai de Hough resultant de la transformada sobre les dades d'un esdeveniment en el detector VELO, mostren un espai de punts en representació de rectes completament saturat, i en el que és pràcticament impossible distingir pics corresponents als punts més votats. Es dedueix que la causa d'aquesta saturació és l'elevat nombre de *hits* amb una densitat molt elevada, ja que l'algorisme de la transformada de Hough pondera totes les parelles de punts per igual, de manera que una parella de *hits* susceptible a formar una recta per la seva direcció, tindrà la mateixa importància que dos *hits* alineats en direcció perpendicular al vèrtex primari. A causa de la densitat de *hits*, el nombre de punts que casualment estan alineats a la projecció és molt gran, encara que gràficament puguin descartar-se clarament com a possible traça, però són comptabilitzats com a tal a causa de la mecànica de l'algorisme.

Es pot concloure doncs, que la informació de les traces queda completament emmascarada per la magnitud de *hits* d'un esdeveniment, i conseqüentment, es descarta la utilització de la transformada de Hough com a únic algoritme per resoldre el problema de reconstrucció de traces a partir de les dades del detector VELO.

Capítol 5

Autòmat Cel·lular

Un cop analitzada la transformada de Hough i els resultats obtinguts amb aquest algorisme aplicat a les dades del detector VELO, es passa a treballar amb un tipus d'algoritme iteratiu, com és l'autòmat cellular, amb l'objectiu de superar els problemes plantejats amb el primer algorisme treballat.

5.1 Introducció i Principi

Un autòmat cel·lular (A.C.) és un model matemàtic dissenyat per un sistema dinàmic que evoluciona en passos discrets. S'utilitza per modelar sistemes que puguin ser discrets, com un conjunt d'objectes simples que interactuen de forma local els uns amb els altres. Inicialment aquests sistemes van ser descoberts dins el camp de la física computacional per John von Neumann a la dècada del 1950 a través del seu llibre “*Theory of Self-reproducing Automata*”[20].

Va ser el mateix von Neumann qui va tenir la idea de crear una màquina amb la capacitat de construir altres màquines a partir de si mateixa i de suportar un comportament complex. Juntament amb Stanislaw Ulam, van implementar la teoria dels autòmats cel·lulars en un vector de dues dimensions $\mathbb{Z} \times \mathbb{Z}$, on \mathbb{Z} representa el conjunt dels enters. Aquest vector el van anomenar espai d'evolucions i , a cadascuna de les seves posicions, cèl·lules. Cada cèl·lula del vector pren un valor inicial d'un conjunt d'estats possibles definit com a $|k| = 29$, i una funció de transició determina el comportament de la mateixa utilitzant el criteri de veïnatge de von Neumann, que consisteix d'un element central $x(i, j)$, anomenat cèl·lula central i les seves cèl·lules veïnes $x(i, j - 1), x(i, j + 1), x(i - 1, j)$ i $x(i + 1, j)$, aquelles cèl·lules més pròximes a dalt, abaix, esquerra i dreta, respectivament [28].

Més endavant, John Horton Conway va popularitzar l'autòmat cel·lular convertint l'algorisme en un joc matemàtic anomenat “*The Game of Life*”, publicat a la revista *Scientific American* l'any 1970[22]. El sistema simula el comportament de cèl·lules活 en un pla bidimensional, i defineix unes regles diferents de veïnatge (veïnatge de Moore) i un comportament específic de la funció de transició.

Al llarg dels anys, s'han realitzat nombroses investigacions sobre el comportament qualitatiu dels A.C. En un d'ells, Stephen Wolfram va observar les diferents evolucions per condicions inicials aleatòries d'A.C. unidimensionals amb únicament dos o tres estats possibles[29]. D'aquesta manera, va determinar una regla que defineix diferents comportaments dels A.C. per diferents condicions inicials, i els va classificar segons el seu comportament qualitatiu en les quatre classes següents:

- **Classe I:** L'evolució acaba de forma estable i homogènia per totes les cèl·lules.
- **Classe II:** L'evolució acaba amb un conjunt d'estructures simples estables o periòdiques.
- **Classe III:** L'evolució porta a un patró caòtic¹.
- **Classe IV:** L'evolució acaba formant estructures de comportament complex, sense ser caòtic ni completament ordenat.

L'aplicació dels autòmats cel·lulars abasta un gran ventall de sistemes diferents que es puguin caracteritzar per tenir un gran nombre de components homogenis i que interactuin de forma local entre si. A la pràctica, qualsevol sistema real al qual puguin aplicar-se els conceptes de “veïnatge”, “estat dels components” i “funció de transició” és candidat a ser modelat per un A.C. Algunes de les àrees d'aplicació on s'utilitzen autòmats cel·lulars són el modelatge de flux de tràfic, vianants i fluids, el modelatge de l'evolució biològica de cèl·lules o de virus o el modelatge de patrons naturals com els presents a la closca del cargol de mar *Conus textile* (figura 5.1).



Figura 5.1: Closca del cargol de mar *Conus textile* amb estructures fractals caracteritzables per un autòmat cel·lular.

¹Es diu d'un sistema complex molt sensible a les variacions sobre les condicions inicials, que impossibilita la prediccio del seu estat a llarg termini.

5.2 Primera Implementació

Amb l'objectiu d'entendre el funcionament dels algorismes basats en autòmats cel·lulars, s'ha implementat dos exemples d'A.C. diferents abans d'aplicar l'algorisme a les dades del detector.

Autòmat Cel·lular Unidimensional

En primer lloc, s'ha començat per la implementació d'un A.C. elemental des de zero. Cal definir quins són els elements clau de l'algorisme: La **graella**. El cas més simple és una graella amb una única línia de cèl·lules, concretament, tal com es pot veure a la figura 5.2, amb 10 cèl·lules.



Figura 5.2: Graella d'un A.C. unidimensional de 10 cèl·lules.

Els **estats**. El conjunt més simple d'estats és binari, amb els estats 0 o 1 (figura 5.3).

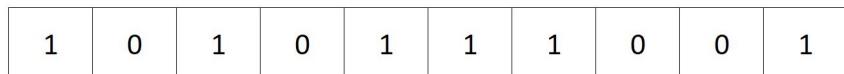


Figura 5.3: Graella d'un A.C. unidimensional de 10 cèl·lules amb estats inicials aleatoris.

Finalment, els **veïns**. En una dimensió el conjunt més simple de veïns per qualsevol cèl·lula és ella mateixa i les dues cèl·lules adjacents, a l'esquerra i a la dreta, tal com mostra la figura 5.4.

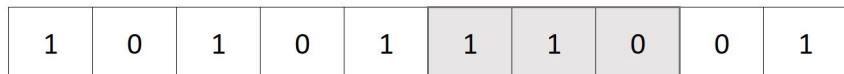


Figura 5.4: Selecció dels veïns de la setena cèl·lula d'un A.C. unidimensional.

Inicialment, es comença amb la línia de cèl·lules amb un estat inicial aleatori i dos veïns per cadascuna. Cal tenir en compte que les cèl·lules primera i última tindran un sol veí, en aquests casos es pot tractar la graella de forma circular, és a dir prenent l'última cèl·lula com a veí de la primera i viceversa, o prenent com a veí una cèl·lula fantasma d'un estat fix.

També cal saber el temps discret en el qual viurà l'A.C., que es defineix com a **generacions**. L'autòmat amb els estats de les condicions inicials correspon a la generació 0.

El pas següent és plantejar la computació de la generació 1 en funció de la 0 i successivament. Per fer-ho, es planteja una fórmula que computa l'estat d'una cèl·lula donat qualsevol temps t [24]:

$$\text{CELL state at time } t = f(\text{CELL neighbors at time } t-1)$$

On CELL equival a qualsevol cèl·lula de la generació següent. Dit d'una altra manera, el nou estat d'una cèl·lula és una funció del seu propi estat, i el de les seves veïnes, a la generació anterior. El pas següent és definir quina és aquesta funció, i el cert és que hi ha moltes maneres diferents de computar aquesta funció de transició. Per exemple, podria calcular-se com el resultat de la suma dels estats de les cèl·lules veïnes o com la mitjana dels seus estats, tal com es fa en tècniques de difuminat pel processament d'imatges digitals. En aquest cas però, s'utilitzarà una tècnica més simple: codificar cadascuna de les possibles combinacions de veïns i cèl·lules amb un estat equivalent segons la figura 5.5.

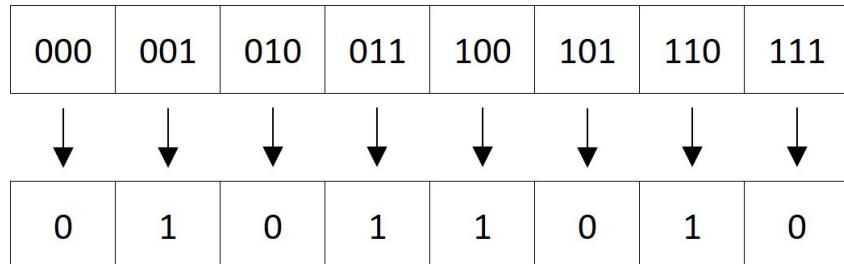


Figura 5.5: Gràfic de la funció de transició utilitzada.

Per poder iniciar la computació de l'autòmat cel·lular únicament queda definir les condicions d'estat inicials. De forma intencionada s'escull començar amb totes les cèl·lules de la generació 0 amb l'estat 0 excepte la central, amb l'estat 1. D'aquesta manera, agrupant les diferents generacions creades s'obté una estructura complexa, tal com es pot veure a la figura 5.6 amb un exemple d'A.C. de 100 cèl·lules.

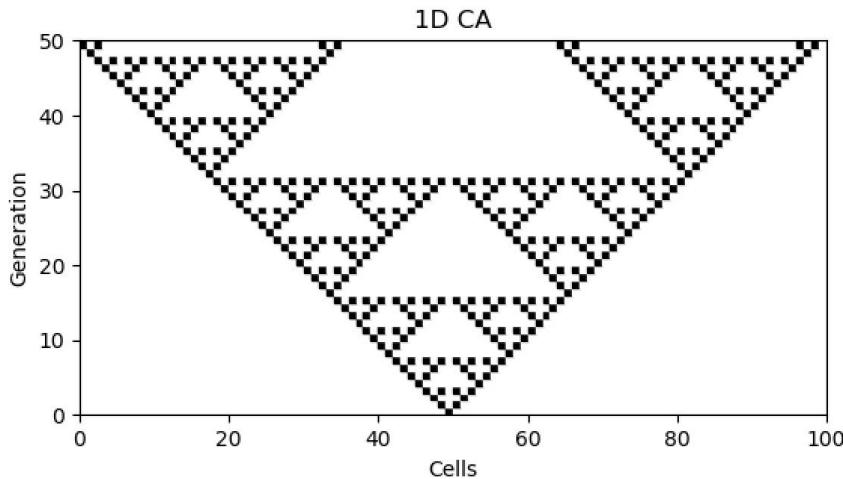


Figura 5.6: Gràfic de l’evolució d’un autòmat cel·lular de 100 cèl·lules al llarg de 50 generacions.

El patró observat correspon a un disseny fractal, una estructura geomètrica que es repeteix a diferents escales, i que, com ja s’ha vist anteriorment, és present a la naturalesa. En cas d’aplicar una funció de transició diferent de l’escollida amb l’exemple, s’obtenen patrons diferents que es poden classificar com a uniformes, repetitius o aleatoris. A la figura 5.7 es pot veure el mateix autòmat cellular utilitzant una funció de transició anomenada regla 190 (la conversió a binari del nombre dóna lloc a la sortida ordenada de la funció de transició per totes les possibles entrades), i on es pot observar un patró repetitiu.

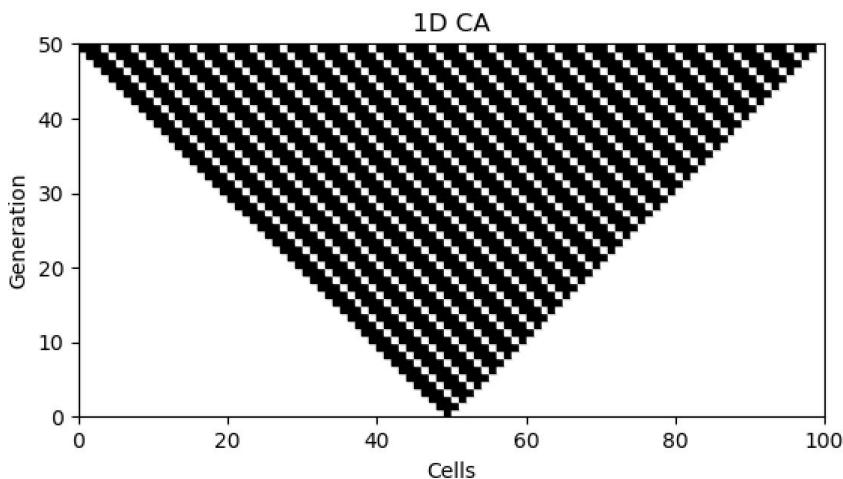


Figura 5.7: Gràfic de l’evolució d’un autòmat celular de 100 cèl·lules al llarg de 50 generacions amb la regla 190.

La figura 5.8 mostra el mateix autòmat cel·lular amb una altra funció de transició, la regla 30, amb la que s'observa un patró aleatori.

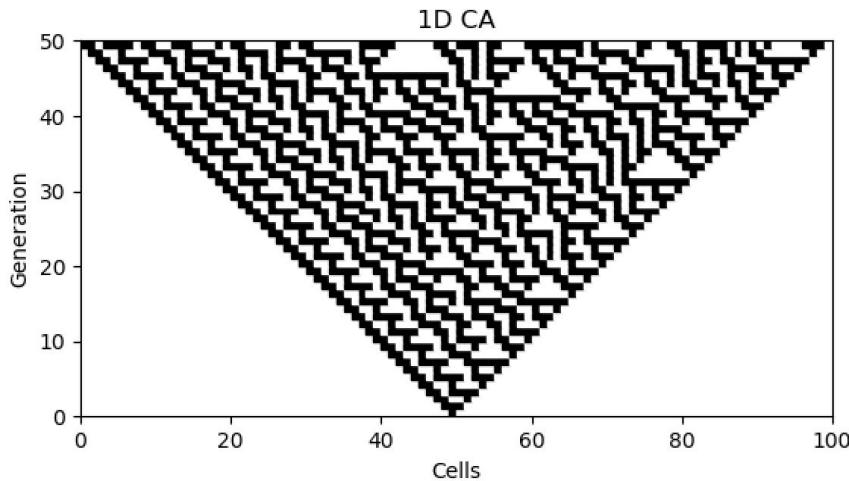


Figura 5.8: Gràfic de l’evolució d’un autòmat cel·lular de 100 cèl·lules al llarg de 50 generacions amb la regla 30.

Un cop implementat l’exemple més simple d’un autòmat cel·lular i vist el seu funcionament més essencial, es passa a implementar un exemple en dues dimensions com és el joc “*The Game of Life*”.

The Game of Life

L’anomenat joc de la vida, va sorgir de la mà del matemàtic britànic John Horton Conway l’any 1970, i va ser catalogat com a “joc” en la mesura que el jugador pot crear una configuració inicial qualsevol i observar l’evolució de l’algorisme, arribant a descobrir certs patrons amb propietats particulars.

L’univers del joc és un espai bidimensional infinit de cèl·lules on cadascuna pot prendre dos estats possibles *viva* o *morta*. Cada cèl·lula interactua amb els seus veïns, definits per les cèl·lules adjacents en vertical, horitzontal i diagonal. A cada generació (temps discret) s’aplica una funció de transició formada per quatre regles:

1. Qualsevol cèl·lula viva amb menys de dos veïns vius mor a causa de **subpoblació**.
2. Qualsevol cèl·lula viva amb dos o tres veïns vius continua viva a la generació següent.
3. Qualsevol cèl·lula viva amb més de tres veïns vius mor a causa de **superpoblació**.
4. Qualsevol cèl·lula morta amb exactament tres veïns vius torna a la vida a causa de la **reproducció**.

A tall de curiositat, al llarg de diverses generacions poden crear-se tot tipus de patrons si més no curiosos. Poden ser estàtics, sense cap canvi a la següent generació, oscil·lants, amb un patró cíclic que retorna a l'estat inicial al cap de diverses generacions, o “naus espacials”, que avancen soles al llarg de la graella. Poden existir també patrons combinats entre si, o simplement patrons caòticos que acabin convergint en alguna combinació concreta, o no.

Passant a la implementació del joc, s'ha acotat a una versió de dimensions simplificades per a poder ser computable. En primer lloc, s'ha definit en la **graella** de cèl·lules com una matriu de dimensions 10×10 amb els marges finits, no circulars. Els **estats** possibles definits com a binaris, de la mateixa manera que en l'exemple unidimensional, representant una cèl·lula *viva* amb l'estat 1 i *morta* amb l'estat 0. Finalment, es defineixen els **veïns** de cada cèl·lula com les vuit cèl·lules adjacents a la seleccionada, com es pot veure a la figura 5.9, formant un total de nou per avaluar amb la funció de transició.

1	0	1	1	1	0	1	0	1	0
0	0	0	0	1	1	0	0	1	0
1	1	1	0	1	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0
0	0	1	1	0	1	0	0	1	1
1	1	1	0	1	0	0	1	0	1
1	1	0	0	1	1	0	0	0	0
1	0	1	1	0	0	1	0	1	0
0	0	1	0	1	1	0	0	1	0
1	1	1	0	1	1	0	1	0	1

Figura 5.9: Graella d'un autòmat cellular bidimensional amb dos estats, ressaltats els veïns de la cèl·lula (7, 6).

En aquest cas, la funció de transició no és tan trivial com en l'exemple unidimensional, ja que les regles són més complexes aquesta vegada. Les configuracions que canvien l'estat d'una cèl·lula depenen del nombre de veïns vius que tingui aquesta al seu voltant i del seu propi estat, per tant, la principal tasca de la funció de transició serà comptar el nombre de veïns vius que té una cèl·lula, viva o morta, al seu voltant, independentment de la seva posició, és a dir, acumular el valor de totes les cèl·lules veïnes i a continuació avaluar, juntament amb el seu propi estat, si es compleix alguna de les quatre regles que determinen un canvi d'estat. A la figura 5.10 es pot veure un exemple de canvi d'estat d'una cèl·lula qualsevol.

<table border="1"><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td></tr></table>	1	0	0	0	1	0	0	0	0	\rightarrow	<table border="1"><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td></tr></table>	1	0	0	0	0	0	0	0	0
1	0	0																		
0	1	0																		
0	0	0																		
1	0	0																		
0	0	0																		
0	0	0																		
<table border="1"><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td></tr></table>	1	0	0	1	0	0	0	0	1	\rightarrow	<table border="1"><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td></tr></table>	1	0	0	1	1	0	0	0	1
1	0	0																		
1	0	0																		
0	0	1																		
1	0	0																		
1	1	0																		
0	0	1																		

Figura 5.10: A l'esquerra, procés de *mort* d'una cèl·lula a causa de subpoblació (un únic veí viu). A la dreta, procés de *naixement* d'una cèl·lula a causa de reproducció (exactament tres veïns vius).

Les condicions inicials que s'han fet servir per posar en funcionament l'algorisme han sigut aleatòries, amb l'objectiu de crear un sistema inicialment caòtic i observar la seva evolució al llarg de cinquanta generacions executades. A continuació es mostren alguns dels resultats obtinguts en diferents implementacions amb una graella de 20×20 per deixar més marge de desenvolupament als possibles patrons.

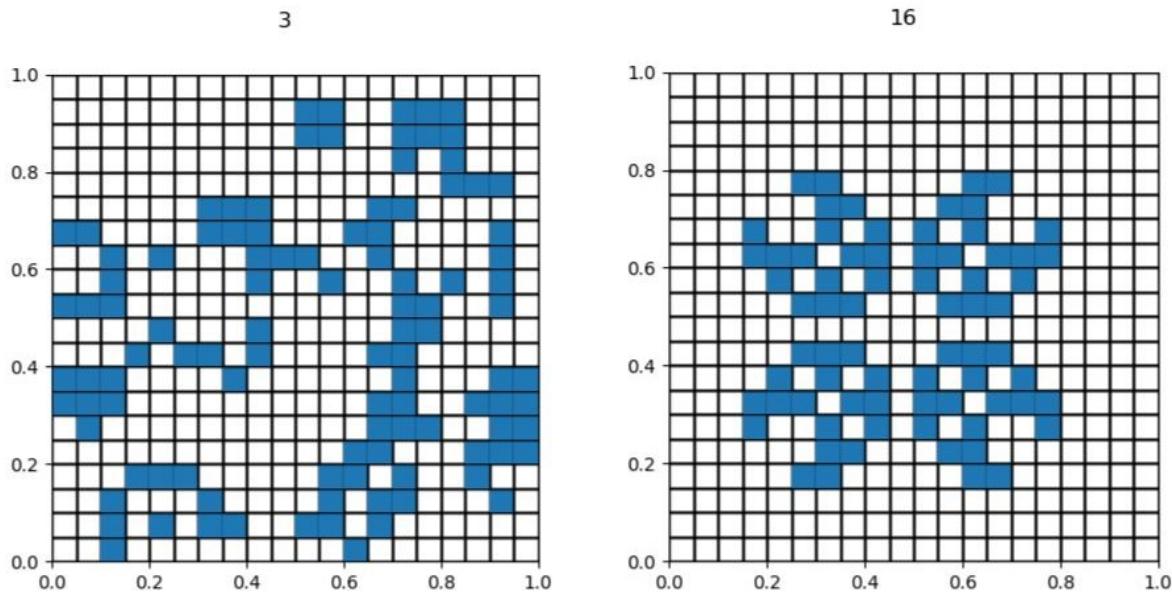


Figura 5.11: A l'esquerra, tercera generació del joc de la vida amb unes condicions inicials aleatòries. A la dreta, un patró cíclic de tres períodes batejat amb el nom de *Pulsar*.

A la figura 5.11 es pot veure un exemple dels patrons caòtics que sorgeixen de les condicions inicials aleatòries amb les properes generacions de l'algorisme. A la imatge de la dreta, a la zona al voltant del punt $(0.5, 0.9)$, s'observa un quadrat format per quatre cèl·lules. Aquesta estructura és un patró estable que es mantindria intacte de forma infinita, però a causa de l'evolució de les cèl·lules adjacents, el patró estable acaba desapareixent o evolucionant a una nova forma. Aquest comportament dóna suport al fet que el Joc de la

Vida és un algoritme no decidable, i que per tant, és impossible construir un algoritme capaç de determinar amb certesa si un patró concret es produirà a partir d'unes condicions inicials.

D'altra banda, a la imatge de la dreta es pot veure un patró oscil·lador cíclic generat espontàniament. A tall d'observació, és curiós observar estructures simètriques i ordenades sorgides a partir d'un patró inicial caòtic i aleatori.

5.3 Adaptació a les Dades Reals

El punt més rellevant dels autòmats cel·lulars, i el que es vol extrapolar a les dades del detector VELO d'LHCb és l'avaluació d'un punt en l'espai en funció dels seus veïns. Aquesta dinàmica és molt interessant per aplicar-la a la cerca de possibles punts que formin una recta, ja que es poden aplicar certes regles, de la mateixa manera que es fa amb el Joc de la Vida, per decidir si dos punts són candidats a formar una recta o no. Aquestes regles es poden resumir de la següent manera:

1. Dos punts són candidats a formar una *seed* si es troben en mòduls adjacents (z i $z+1$) i la distància tridimensional entre ells és inferior a 0.4.
2. Un punt és candidat a formar part d'una recta si, al fer una predicció de les coordenades x i y , projectant la recta sobre el mòdul z al qual pertany el punt a avaluar, aquesta distància és menor a una tolerància de 0.7 tenint en compte un coeficient de dispersió² de 0.4.

Tenint present un sistema tridimensional, ja que les dades del detector corresponen a un espai de tres dimensions, es vol avaluar cadascun dels punts, que representen els *hits*, com a cèl·lules d'un autòmat, observant els seus veïns, els *hits* més pròxims, i ponderar aquells que compleixin les regles anteriors i siguin candidats a formar part d'una traça d'una partícula que ha passat pel detector.

Per procedir a una implementació d'aquest tipus, cal tenir present el format de les dades un cop processades per l'*event model*, segons s'explica al capítol 3. És a dir, la informació de la qual es disposa difereix en gran mesura d'una estructura matricial idònia per aplicar un algorisme de cerca de veïns, com és un autòmat cel·lular. Idealment, es voldria tenir una estructura tridimensional, on cada casella representés un punt del VELO susceptible a detectar el pas d'una partícula, amb un valor 1 o 0 en cas afirmatiu o negatiu respectivament. En aquest suposat cas, la identificació dels *hits* més pròxims a qualsevol punt del sensor es converteix en un problema trivial. Malauradament, de la mateixa manera que s'ha trobat amb la implementació de la transformada de Hough, una estructura matricial de les dimensions requerides per emmagatzemar tots els punts del detector VELO no és computable.

Per tant, es passa a buscar una forma alternativa de buscar *hits* veïns a partir del model estructural d'un objecte *event*.

²Fenomen físic que modifica les trajectòries de les partícules a causa d'irregularitats en el medi que travessen.

Cos de l'Algorisme

Treballant amb les dades de sortida del detector VELO, únicament es disposa d'un conjunt de *hits*, cadascun amb les seves coordenades, però sense cap relació els uns entre els altres. De manera que, a priori, és impossible saber quins *hits* estan pròxims entre si per establir uns llindars de veïnatge. Per fer-ho, s'hauria de mesurar la distància entre el *hit* a avaluar i tota la resta, per poder identificar quins són els seus veïns més pròxims. I seguidament, repetir aquesta operació per cadascun dels *hits* d'un esdeveniment. Una implementació amb aquesta dinàmica implica un cost computacional $O(N)$, on N fa referència al nombre de *hits* en un esdeveniment, el qual pot variar al voltant de 1000 fins a 4500 aproximadament en el conjunt de dades que es disposen, esdevenint un cost insostenible tenint en compte que es requereix una cerca de veïns per cada *hit* d'un esdeveniment.

De tota manera, tal com està definit l'objecte `event`, es disposa de tots els *hits* classificats en funció del sensor on es troben. Aquest fet comporta una reducció considerable de l'espai de cerca de veïns, ja que, tal com s'ha determinat prèviament, dos punts seran candidats a formar una recta si es troben en sensors adjacents, és a dir, els veïns susceptibles a ser candidats a formar una recta amb el *hit* situat al sensor z , es trobaran al sensor $z + 1$ (figura 5.12).

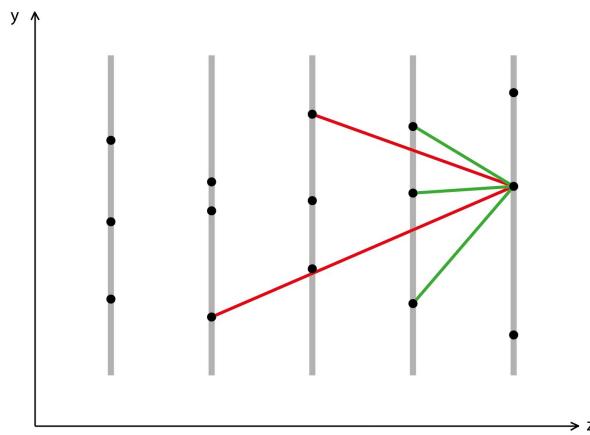


Figura 5.12: Representació del procés de cerca de veïns amb les dades del detector VELO. Les parelles de punts unides per una línia vermella, segons la norma, no seran mai veïns. Les parelles de punts unides per una línia verda, són susceptibles a formar una recta.

Amb aquesta premissa, el cost computacional de l'algoritme es redueix a $O(M)$ amb M equivalent al nombre de *hits* del sensor $z + 1$. I així és com s'ha fet la implementació d'un algoritme de reconstrucció de traces a partir d'aquest principi, *Tracking Wheels* [23].

Tracking Wheels

El procés de reconstrucció d'aquest algoritme comença observant els últims dos mòduls del detector (correspondents als sensors 51, 50, 49 i 48), amb l'objectiu de comparar tots els *hits* de l'últim mòdul (sensors 51 i 50) amb tots els del penúltim (sensors 49 i 48) i decidir, en funció de la distància en *x* i *y* entre dos punts, si aquests són susceptibles a formar una *seed*. Per fer-ho, s'utilitza la funció següent:

```
def are_compatible(self, hit_0, hit_1):
```

On es compara la distància màxima dx/dz i dy/dz amb la distància entre *hit_0* i *hit_1*. Si el retorn d'aquesta funció és `True`, vol dir que *hit_0* i *hit_1* formen l'inici d'una traça. A partir d'aquí, es procedeix a comparar la recta que formen els dos punts anteriors amb tots els *hits* dels tres següents mòduls (en el sentit de les *z* negatives). Per fer-ho, s'utilitza la següent funció:

```
def check_tolerance(self, hit_0, hit_1, hit_2):
```

On *hit_0* i *hit_1* són els punts que formen la *seed* trobada com a compatible i *hit_2* és el nou punt a avaluar. En aquesta funció es fa una predicció, de les coordenades *x* i *y* que hauria de tenir el punt *hit_2* a partir del punt on es talla amb la recta formada per la *seed* amb la *z* corresponent al mòdul on es troba el *hit_2*. Si la diferència en *x* i en *y* entre el punt predit i les coordenades del *hit_2* són menors a un paràmetre de tolerància màxima, es determina el conjunt dels tres punts com a una recta inserint-los en una nova *track*.

A continuació, es procedeix a avaluar els *hits* de la resta de mòduls del detector amb el mateix procediment, passant com a paràmetres de la funció de tolerància els dos últims *hits* inserits a la *track* i el *hit* a comparar.

En el cas en què no es trobi cap *hit_2* compatible amb la *seed* original, aquesta és descartada i es procedeix a buscar noves *seeds* amb altres punts d'origen.

Finalment, si s'ha trobat una traça compatible, un cop comparats tots els *hits* del mòdul 0, es classifica la traça formada en funció del nombre de *hits* que la formen. Si conté quatre o més *hits*, és fortament susceptible a ser una traça ben reconstruïda, així doncs, aquesta s'inclou a una llista de traces (**tracks**) i també s'inclouen els seus *hits* a una llista anomenada **used_hits**, que serveix per descartar de la cerca tots aquells punts que ja han estat utilitzats en una traça. Si, per contrari, el nombre de *hits* que forma la traça trobada és igual a tres, és catalogada com una traça dèbil, i conseqüentment, susceptible a ser soroll o una *clone track*, per tant, s'inclou a una llista anomenada **weak_tracks** sense comptabilitzar els seus *hits* com a usats.

No és fins un cop acabat tot a la cerca de traces de tots els punts del detector, que es processen les **weak_tracks**. En aquest moment, es comprova si algun dels *hits* que formen aquestes traces pertany a la llista **used_hits**, en cas afirmatiu, la traça és descartada, i en cas negatiu, la traça és inclosa a la llista **tracks**.

5.4 Resultats

Amb l'algorisme descrit anteriorment, s'aconsegueixen uns molt bons resultats respecte a la precisió de la reconstrucció de les dades comparades amb les simulacions Monte Carlo dels esdeveniments. Com a exemple es presenten dues taules que resumeixen els resultats d'un esdeveniment, `event_0`, a la taula 5.1. L'eficiència de reconstrucció, en aquest cas és màxima tot i que tant la fracció de *clones* com la fracció de *ghosts* no són exactament 0 com en el cas ideal.

Number of Hits:	1003
Found Tracks:	117
Efficiency, [0 – 1]:	1.0
Clone Fraction, [0 – 1]:	0.066666
Ghost Fraction, [0 – 1]:	0.025641

Taula 5.1: Resultats generals de la reconstrucció de l'esdeveniment 0.

Track Type	Reconstruction Rate	Clone Rate	Purity	Hit Efficiency
VELO	93.3%	1.87%	98.99%	98.01%
Long	100%	5.13%	97.98%	96.74%
Long > 5 GeV	100%	6.67%	98.03%	96.45%
Long Strange	100%	0.0%	100%	100%
Long Strange > 5 GeV	None	None	None	None
Long Fromb	100%	0.0%	98.46%	98.48%
Long Fromb > 5 GeV	100%	0.0%	100%	100%

Taula 5.2: Resultats de la reconstrucció de l'esdeveniment 0, desglossat segons el tipus de traces trobades.

A la taula 5.2 es mostren els resultats de l'algorisme de manera més extensa, desglossant amb l'eficiència, factor de *clone tracks*, puresa de les traces i eficiència dels *hits* dels diferents tipus de traces que es troben. Gràcies al fet que els esdeveniments disponibles han estat generats per una simulació Monte Carlo, a banda de les dades de sortida dels sensors, també es disposa de la informació de les traces reconstruïdes, de les respectives partícules de procedència i també de l'energia d'aquestes. Així doncs, és possible calcular l'eficiència de cada tipus de traça per separat, en funció de la partícula de procedència i de l'energia d'aquesta.

La puresa d'una traça es defineix com la proporció de *hits* que pertanyen realment a la traça Monte Carlo associada a una partícula en concret, respecte a tots els *hits* de la traça reconstruïda. El valor ideal de puresa és el definit com a màxim per construcció i correspon al valor de 100%. L'eficiència de *hits* és un valor complementari a la puresa i es calcula com la relació entre el nombre de *hits* d'una traça reconstruïda que pertanyen a la traça

MC d'una partícula entre el nombre de *hits* totals associats a la partícula, de manera que el seu valor ideal i també màxim és del 100%. Mentre que la puresa dóna una magnitud de la quantitat de soroll afegit a una traça reconstruïda, l'eficiència de *hits* dona una magnitud de la quantitat de *hits* reals que s'han associat correctament a una traça d'una partícula concreta.

Cal tenir en compte que no tots els esdeveniments tenen el mateix nombre de *hits* ni el mateix nombre de trases, fet que pot variar la complexitat de l'algorisme així com els paràmetres anteriors o el temps emprat en la reconstrucció. Així doncs, per avaluar el comportament de l'algoritme s'ha comparat l'eficiència, la taxa de *clone tracks* i la taxa de *ghost tracks* per als trenta esdeveniments diferents disponibles. La gràfica resultant es pot veure a les figures 5.13, 5.14 i 5.15.

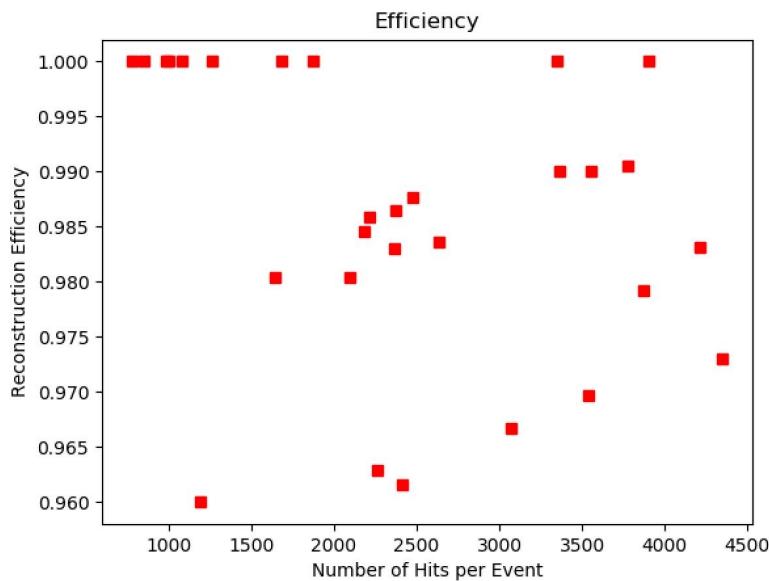


Figura 5.13: Gràfica de l'eficiència de l'algoritme de reconstrucció en funció del nombre de *hits* per esdeveniment.

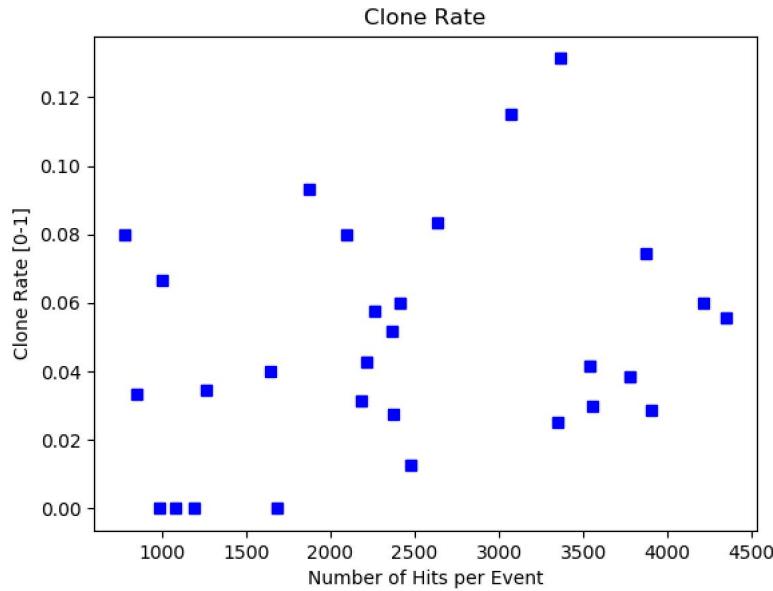


Figura 5.14: Gràfica del nombre de *clone tracks* reconstruïdes en funció del nombre de *hits* per esdeveniment.

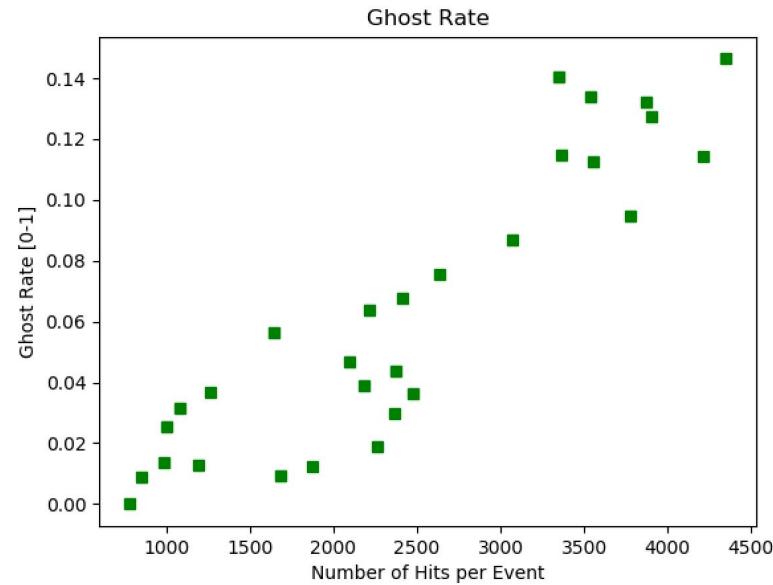


Figura 5.15: Gràfica de la taxa de *ghost tracks* reconstruïdes en funció del nombre de *hits* per esdeveniment.

De les gràfiques anteriors, es pot extreure que no hi ha una relació clara entre l'eficiència de reconstrucció i el nombre de *hits* dels esdeveniments, així doncs, es pot assegurar que

l'eficiència no pren una dependència important amb el nombre de *hits* i que per tant no varia en excés amb diferents esdeveniments. De la mateixa manera succeeix amb la reconstrucció de les *clone tracks*, no s'observa una relació clara amb el nombre de *hits* i la variació del paràmetre, però sí que es pot observar, a la gràfica 5.15, un petit augment de la taxa de *ghost tracks* a mesura que s'augmenta el nombre de *hits* dels esdeveniments.

5.5 Temps d'Execució

D'altra banda, l'objectiu principal dels algoritmes de reconstrucció d'LHCb no és únicament tenir la millor eficiència possible, sinó fer-ho en el menor temps possible, ja que les especificacions de l'actualització del detector requereixen una velocitat de presa de dades de cada 25ns . És interessant doncs, avaluar el temps d'execució d'aquest algorisme sobre un *hardware* del qual es coneixen les especificacions i observar el comportament enfront esdeveniments amb diferent nombre de *hits*.

Especificacions de Maquinari

El maquinari sobre el qual s'ha executat l'algoritme presenta les especificacions de la taula 5.3. El mateix maquinari serà utilitzat en totes les execucions amb les condicions més similars possibles amb l'objectiu de normalitzar els resultats de temps mesurats i fer-los comparables.

Intel Core i7 6600U		
CPU	Cores	2
	Threads	4
	Caché	4MB
	Frequency	2.5 GHz
RAM	Size	16256 MB
Graphics	Intel HD Graphics 520	
Storage	Samsung PM951 (NVMe) SSD	
	Size	512 GB

Taula 5.3: Especificacions del *hardware* utilitzat per executar els algoritmes.

D'altra banda, totes les implementacions s'han realitzat amb el llenguatge Python 3.6 [14] i sobre l'IDE (*Integrated Development Environment*) de JetBrains PyCharm.

Mesures de Temps

Per avaluar el comportament de l'algorisme quant a temps d'execució, s'han realitzat mesures del temps emprat en extreure les traces reconstruïdes a partir de les dades processades dels sensors per tots els esdeveniments disponibles. Per fer aquesta mesura s'ha utilitzat la

llibreria de Python `time` [14], que mitjançant la següent comanda, es retorna el temps de CPU del sistema utilitzat pel procés actual en fraccions de segon.

```
time.process_time()
```

Amb l'objectiu de normalitzar les mesures obtingudes respecte a l'ocupació de la CPU, s'han realitzat cent mesures de temps sobre el mateix esdeveniment, és a dir, sobre la mateixa execució, i s'ha pres com a resultat de la mesura la mitja dels anteriors, donant com a resultat final la gràfica de la figura 5.16.

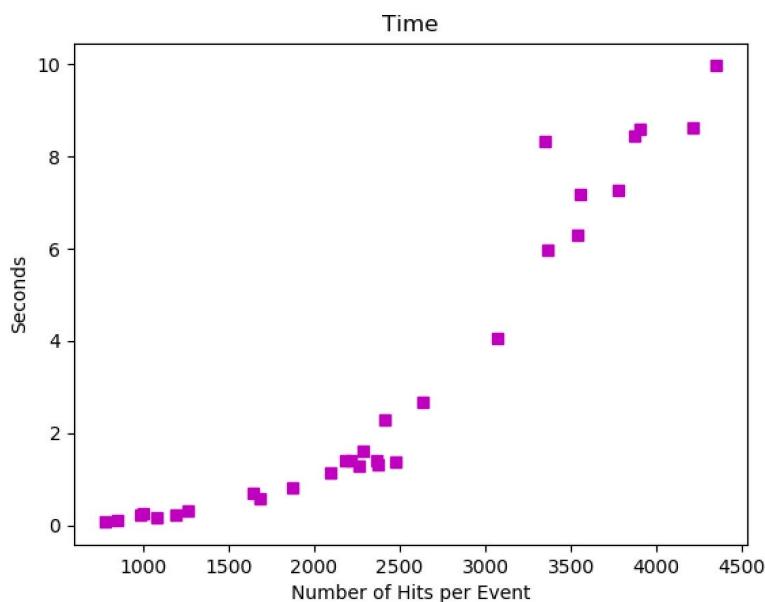


Figura 5.16: Gràfica del temps d'execució de l'algoritme, sobre el *hardware* especificat, en funció del nombre de *hits* dels diferents esdeveniments.

Els resultats obtinguts en temps de l'execució de l'algoritme mostren una clara tendència de creixement exponencial a mesura que augmenta el nombre de *hits* per esdeveniment. Aquest fet es deu a la utilització d'estructures de dades lineals en la implementació del mateix, en concret, per implementar la cerca dels *hits* més pròxims sobre iteracions de llistes de punts. Si per cadascun dels *hits* d'un esdeveniment s'ha de buscar el *hit* més pròxim d'entre una llista de punts del sensor adjacent, a mesura que el nombre de *hits* d'un esdeveniment augmenti, el nombre d'elements de la llista també augmentarà, i el temps emprat en realitzar una iteració per tots els elements de la llista esdevé una variable exponencial.

Tenint present que un dels objectius de l'actualització del detector i de l'experiment és augmentar la quantitat de dades que es prenen, és interessant contemplar estructures de dades alternatives, que no presentin una dependència tan gran amb la quantitat de dades

pel que fa al cost computacional de cada execució, però que, a la vegada, permetin utilitzar la mateixa estructura d'execució per mantenir els mateixos paràmetres d'eficiència.

Capítol 6

Procés d'Optimització de l'Algoritme

Tot i que els resultats obtinguts amb la implementació anterior d'un algorisme basat en un autòmat cel·lular han estat molt bon respecte la precisió i eficiència de la reconstrucció, el cost de l'algorisme segueix tenint un cost computacional exponencial que pot arribar a suposar un problema de temps en augmentar considerablement el nombre de *hits* d'un esdeveniment. Amb l'objectiu de millorar aquest aspecte de l'anterior algoritme, en aquest capítol es buscarà una estructura de dades que permeti reduir el temps de cerca de *hits* pròxims a cada iteració.

6.1 Introducció

Observant les dades del detector projectades sobre el pla XY , tal com s'ha mostrat anteriorment a la secció 4.4 amb la imatge A de la figura 4.10, de manera gràfica, és fàcil identificar aquells *hits* que formen part d'una mateixa traça, ja que estan situats en mòduls adjacents i alineats amb un punt de fuga al centre de la imatge. Si, en canvi, observem únicament una part de les dades, en concret, els *hits* dels dos últims mòduls del detector en el mateix esdeveniment, projectats de la mateixa manera sobre el pla XY , com en la figura 6.1, són fàcilment identifiables aquelles parelles de punts, de mòduls adjacents (de diferent color), que formen l'inici d'una recta, o el que s'anomena *seed*.

De la mateixa manera, es veu a simple vista, que fer la comprovació de distància entre els *hits* $h1$ i $h5$ és absurda, ja que, tot i que pertanyen a mòduls adjacents, gràficament s'observa que la distància entre ells queda fora del domini de veïnatge d' $h1$. En canvi, sí que té sentit que es comprovi la distància entre $h2$ i $h2$, $h3$ i $h4$, ja que aquests són els que gràficament es troben més a prop i són els millors candidats a formar una traça juntament amb $h1$.

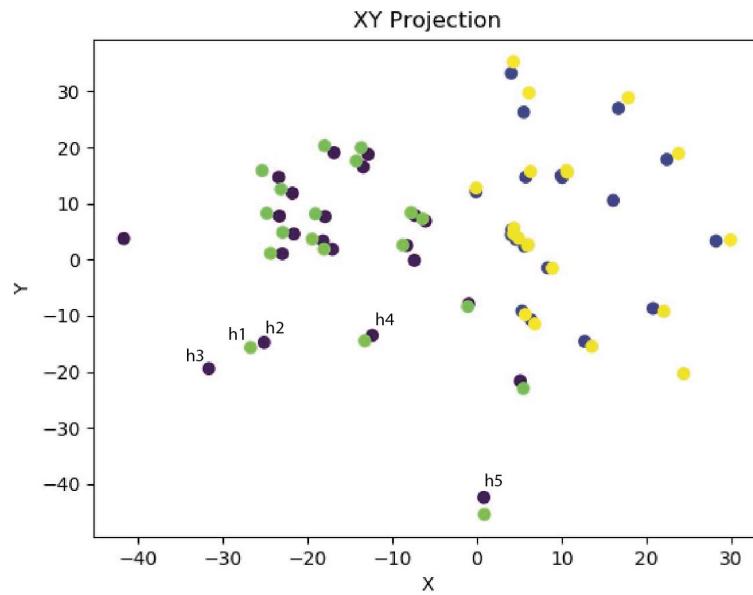


Figura 6.1: Projecció sobre el pla XY dels *hits* dels sensors 48, 49, 50 i 51 del detector VELO.

Ja s'ha pogut comprovar que aquesta dinàmica tan simple des d'un punt de vista gràfic no és trivial d'implementar amb estructures de dades lineals, de manera que s'ha buscat una estructura no lineal capaç d'organitzar les dades en funció de les seves coordenades i que estableixi relacions de proximitat entre elles amb l'objectiu de reduir l'espai de cerca de veïns de cadascun dels *hits* i conseqüentment, el temps d'execució de cada iteració.

Una estructura de dades que compleix els criteris exposats és l'arbre *R-Tree*.

6.2 Introducció a l'*R-Tree*

Un arbre, en termes d'informàtica, és un tipus abstracte de dades que imita l'estructura jeràrquica d'un arbre, amb un valor a l'arrel i diversos sub-arbres, cadascun amb un node pare, representats com un conjunt de nodes entrelaçats. Un node d'un arbre és una estructura de dades amb un valor i una llista de referències als seus propis nodes (els fills), amb la condició que no hi hagi cap referència duplicada ni que cap node apunti a l'arrel. La figura 6.2 mostra un exemple d'un arbre simple sense ordenar, on el node amb l'etiqueta 2 és l'arrel, sense cap pare, i té dos fills, el 7 i el 5, cadascun amb els seus respectius fills.

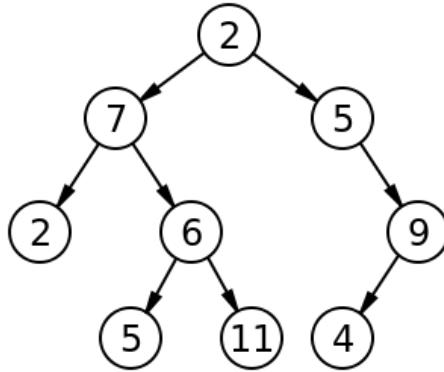
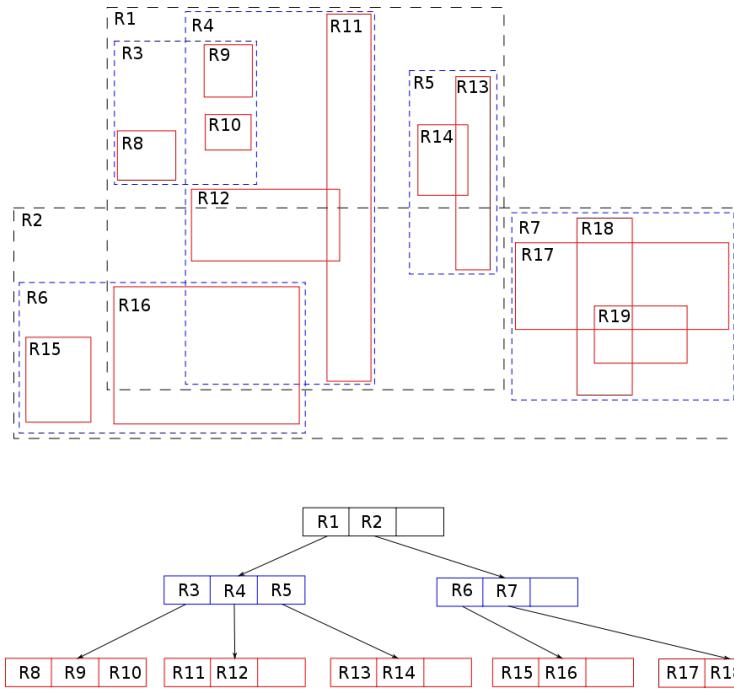


Figura 6.2: Arbre simple sense ordenar.

Existeixen moltes tipologies diferents d'arbres, cadascun amb unes propietats determinades. Un d'ells és l'anomenat *Region-Tree* o *R-Tree*. Els *R-Trees*, proposats l'any 1984 per Anthonin Guttman, són estructures de dades utilitzades per mètodes d'accés espacial, és a dir, per indexar informació multi-dimensional, com per exemple, coordenades (x, y) d'un lloc geogràfic. La seva estructura es basa en dividir l'espai de dades de forma jeràrquica en conjunts que poden, o no, estar superposats, agrupant aquells objectes que es troben més a prop en un quadrat amb el mínim perímetre. La figura 6.3 en mostra un exemple.

Figura 6.3: Exemple de l'estrucció d'un *R-Tree* i representació de l'organització de cadascun dels seus nodes.

Els *R-Trees* són semblants als *B-Trees*, ja que ambdós són balancejats, és a dir, que tots els nodes fulla¹ estan com a molt a un nivell de distància dins de l'arbre. Respecte al disseny de les dades, l'*R-Tree* s'organitza en pàgines, que poden tenir un nombre variable d'entrades predeterminades. Cada entrada guarda dues dades diferents: un identificador del node fill i el rectangle conjunt límit de totes les entrades d'aquell node fill. Cadascun dels nodes fill guarda la informació corresponent a la dada que representa, pot ser un punt o una forma poligonal en l'espai.

A banda dels principals algoritmes de cerca presents a la gran majoria d'arbres, com per exemple inserció i contingència, l'*R-Tree* presenta un algorisme de cerca del veí més proper o *nearest neighbor search* (NNS), que resol el problema d'optimització de trobar el punt o conjunt de punts més propers a un punt determinat. També consta d'una cerca per intersecció, en la que busca, dins del conjunt de dades, tots aquells punts que tallin amb una àrea donada. Totes les operacions de cerca tenen un cost computacional $O(\log_M N)$ on M és el nombre d'entrades de cada node i N el nombre de nodes de l'arbre.

6.3 Aplicació de l'*R-Tree* a l'Algorisme de Reconstrucció

Com ja s'ha comentat a l'apartat 6.1, es vol utilitzar l'estruatura de dades d'un *R-Tree* amb l'objectiu de reduir l'espai de cerca dels *hits* més pròxims entre si per determinar de forma és ràpida aquelles parelles susceptibles a formar una traça. Per fer-ho, s'ha utilitzat la llibreria de Python “Rtree” [15], que proporciona diverses funcions d'indexació espacial avançades. Aquestes són: cerca de veïns propers, cerca d'intersecció, índexs multi-dimensionals, índexs en *clusters*, càrregues de gran volum, supressió, serialització del disc i implementació personalitzada d'emmagatzematge.

Respecte a la implementació de l'*R-Tree* dins de l'algorisme de reconstrucció basat en l'autòmat cellular, cal tenir present que únicament caldrà construir un arbre per esdeveniment, ja que les accions que es realitzaran al llarg de l'execució del mateix seran únicament cerques. En primer lloc, cal crear l'estruatura de l'arbre a utilitzar, amb la particularitat que l'espai de dades creat sigui de tres dimensions. Es declara doncs, l'índex de l'*R-Tree* de la següent manera:

```
p = index.Property()
p.dimension = 3
idx3d = index.Index(properties=p, interleaved=False)
```

A continuació, s'han d'inserir a l'arbre les dades pertinents, és a dir, la informació corresponent a cada *hit* de l'esdeveniment. S'utilitza com a identificador d'entrada el propi identificador de cada *hit* i les coordenades (x, y, z) com a posició espacial del *hit*. Cal

¹Aquell node d'un arbre que té un pare però cap fill.

tenir present que també és possible inserir rectangles a l'arbre en comptes de punts, de manera que les coordenades de la funció d'inserció vénen determinades per tres valors ($x_{max}, x_{min}, y_{max}, y_{min}, z_{max}, z_{min}$). Doncs, per inserir un sol punt cal donar el mateix valor a les coordenades mínimes i màximes. Addicionalment, una entrada pot guardar un objecte de dades associat a l'identificador i les coordenades anteriors. Per simplificar l'extracció de dades de cada *hit* en els processos de cerca posteriors, s'insereix per cada entrada, un objecte `hit` amb les corresponents dades. S'insereixen doncs, les dades de forma següent per cadascun dels *hits* de l'esdeveniment:

```
idx3d.insert(h.id, coordinates=(h.x, h.x, h.y, h.y, h.z, h.z),
    ↳ obj=hit(h.x, h.y, h.z, h.id, h.sensor_number))
```

A partir d'aquest moment, únicament cal realitzar consultes de cerca de veïns a l'arbre creat, mitjançant la funció següent:

```
def nearest(self, coordinates, num_results=1, objects=False):
```

En la que es passen com a paràmetres les coordenades, `coordinates`, del punt respecte al que es vol trobar els veïns més pròxims, un nombre enter per indicar el nombre de veïns a retornar, `num_results` i un *flag* indicant si es retorna l'objecte associat a l'entrada corresponent, juntament amb l'identificador de l'entrada retornada, `objects`.

A causa de la geometria del detector, la separació entre els mòduls de sensors adjacents (la diferència de la posició z) és força gran en comparació amb la diferència màxima en els eixos x i y acceptable per considerar dos *hits* com a *seed*. De manera que si les coordenades que donem a la funció `nearest` són exactament les de *hit* respecte al que es busquen veïns (*hit* origen), és molt possible que els identificadors retornats per la funció corresponguin a *hits* sobre el mateix mòdul z que l'origen, ja que aquests físicament seran més pròxims que els *hits* del mòdul $z_{origen} + 1$. Tenint aquest fet en compte, les coordenades que s'introdueixen a la funció `nearest` corresponen a la projecció del *hit* origen sobre el pla $z_{origen} + 1$, es mostra un exemple a la figura 6.4, d'aquesta manera s'assegura que els *hits* més pròxims retornats per la cerca es trobaran al mòdul que interessa. De manera que l'operació de cerca es crida de la següent manera:

```
neighbors = list(idx3d.nearest((h0.x, h0.x, h0.y, h0.y, s1_z, s1_z),
    ↳ 3, objects=True))
```

De manera que es retorna dins la variable `neighbors`, una llista d'objectes amb les dades corresponents a cadascun dels, en aquest exemple, 3 *hits* més pròxims a l'origen (*h0*). El nombre de punts a retornar és un paràmetre discutit a l'apartat de resultats 6.8.

Aquesta operació de cerca s'executa reiteradament al llarg de l'algoritme, en les mateixes ocasions en les quals en l'algoritme original es recorria a una cerca iterativa per tots els *hits* del respectiu mòdul adjacent.

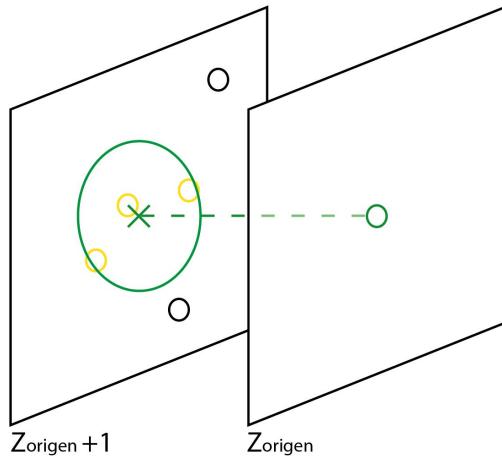
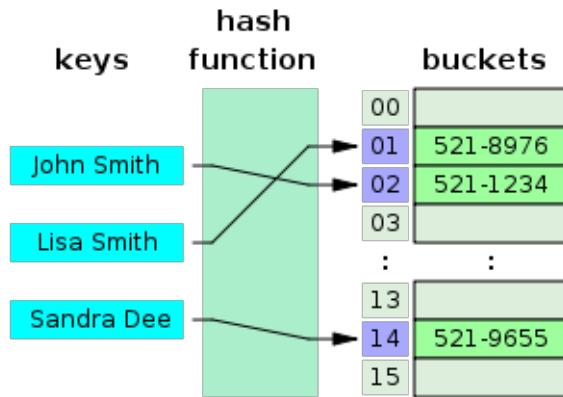


Figura 6.4: Esquema de la projecció d'un *hit* situat al mòdul z_{origen} , en color ver, sobre el mòdul $z_{origen} + 1$, per trobar els seus *hits* més pròxims del mateix mòdul, marcats en groc.

6.4 Taules de *Hash*

Una taula de *hash*, també anomenada matriu associativa o taula de dispersió, és una estructura de dades que associa claus amb valors. S'utilitza principalment per fer cerques de manera eficient, ja que permet l'accés als elements guardats a partir d'una clau generada. Per exemple, es pot accedir a les dades de telèfon i direcció utilitzant una clau generada amb un nom o un número de compte. El seu funcionament es basa a transformar la clau mitjançant una funció de *hash*, en un número que identifica la posició de la taula de *hash* on es troba la informació de l'element associat a la clau. A la figura 6.5 es mostra un esquema de funcionament.

D'altra banda, les taules de *hash* presenten una particularitat molt interessant: el temps de cerca és constant i de cost $O(1)$ independentment del nombre d'elements de la taula. Tot i que es pot donar el pitjor cas en qual el cost de cerca sigui $O(N)$ on N fa referència al nombre d'elements de la taula, en aquesta situació, el cost seria el mateix que una cerca en una estructura de tipus llista.

Figura 6.5: Esquema de funcionament d'una taula de *Hash*.

6.5 Aplicació de taules de *Hash* a l'Algorisme de Reconstrucció

Tal com s'ha explicat a l'apartat 5.3, a la implementació de l'algoritme de reconstrucció basat en un autòmat cel·lular, s'utilitza de recurs una llista, amb el nom de `used_hits`, per comptabilitzar aquells *hits* que ja han estat reconstruïts en una traça amb l'objectiu de no tenir-los en compte a les següents iteracions. La utilització d'aquest recurs es fa mitjançant la següent comanda, que retorna `True` en cas que l'identificador `id` es trobi llistat a `used_hits`, o `False` en cas contrari.

```
if id not in used_hits:
```

Tot i que la comanda és simple, la seva execució interna implementa una cerca de l'element `id` dins la llista, aquest fet implica una consulta de tots els elements inserits a `used_hits` comparant-los amb l'element que es vol trobar, esdevenint en un cost computacional $O(N)$ on N , el nombre d'elements, es va incrementant a mesura que avança l'execució de l'algorisme a causa de l'addició de noves traces reconstruïdes a la llista.

Per aquest motiu, s'ha optat per utilitzar una taula de *hash* que guardi com a elements els *hits* que ja han estat reconstruïts, de manera que la comprovació d'existència d'un nou *hit* a la llista sigui sempre amb un cost $O(1)$.

La implementació d'una taula de *hash* en Python és simple, s'utilitza a través d'una estructura anomenada diccionari, que internament s'implementa com una taula de *hash*, i es declara de la següent manera:

```
used_hits = {}
```

Ja que l'objectiu d'aquesta taula de *hash* és fer una comprovació ràpida de la utilització d'un *hit*, es defineix l'estructura del diccionari amb l'identificador de *hit* com a clau, i un booleà com a element associat, que indica si aquell *hit* ja ha estat reconstruït (`True`) o no (`False`). Però a la pràctica, no és necessari inserir a priori tots els *hits* al diccionari amb el corresponent element a `False`, ja que es pot consultar únicament si un identificador de *hit* ha estat inserit a la taula amb la comanda `not in`. D'aquesta manera, l'única inserció que es fa al diccionari és en el moment que es considera una traça completament reconstruïda. Així doncs, els *hits* reconstruïts s'insereixen al diccionari amb el valor `True` amb la següent comanda:

```
used_hits[h.id] = True
```

6.6 Resultats

Després de la implementació de l'estructura de dades de l'arbre *R-Tree* i la taula de *hash* sobre l'algorisme de reconstrucció de trases basat en l'autòmat cel·lular, s'ha avaluat la seva eficiència i el seu comportament en primera instància sobre tots els esdeveniments disponibles, de la mateixa manera que l'anàlisi fet per l'algorisme original.

A les taules 6.1 i 6.2, es poden veure els resultats desglossats de les diferents trases reconstruïdes del mateix esdeveniment que el mostrat a les taules 5.1 i 5.2.

Number of Hits:	1003
Found Tracks:	126
Efficiency, [0 – 1]:	1.0
Clone Fraction, [0 – 1]:	0.0
Ghost Fraction, [0 – 1]:	0.103174

Taula 6.1: Resultats generals de la reconstrucció de l'esdeveniment 0 amb l'algoritme optimitzat.

Comparant les taules anteriors amb les de l'algorisme sense optimitzacions, es pot veure que la precisió ha augmentat notablement amb la reducció de la taxa de *clone tracks*, tot i que, per contra, la taxa de *ghost tracks* també ha augmentat. Però analitzant els resultats de la taula 6.2, els paràmetres obtinguts quant a eficiència de *hits*, no són coherents, ja que, com s'ha definit a l'apartat 5.4, per construcció, el càlcul d'eficiència de *hit* no pot ser superior al 100%, fet que indicaria que a la traça reconstruïda s'han trobat més *hits* pertanyents a

la partícula que al propí Monte Carlo. Es conclou que hi ha un error de càlcul a l'eina de validació utilitzada que dóna uns resultats incoherents en aquest cas.

Track Type	Reconstruction Rate	Clone Rate	Purity	Hit Efficiency
VELO	93.9%	0.00%	100%	113.78%
Long	100%	0.00%	100%	115.35%
Long > 5 GeV	100%	0.00%	100%	110.34
Long Strange	100%	0.00%	100%	170.05
Long Strange > 5 GeV	None	None	None	None
Long Fromb	100%	0.0%	100%	102.30
Long Fromb > 5 GeV	100%	0.0%	100%	102.49

Taula 6.2: Resultats de la reconstrucció de l'esdeveniment 0 amb l'algoritme optimitzat, desglossat segons el tipus de traces trobades.

De la mateixa manera que s'ha fet amb els resultats de l'anterior algorisme, per poder avaluar el comportament d'aquest en diferents esdeveniments, s'ha realitzat una gràfica per als paràmetres d'eficiència, *clone rate* i *ghost rate* en funció del nombre de *hits* de tots els esdeveniments disponibles.

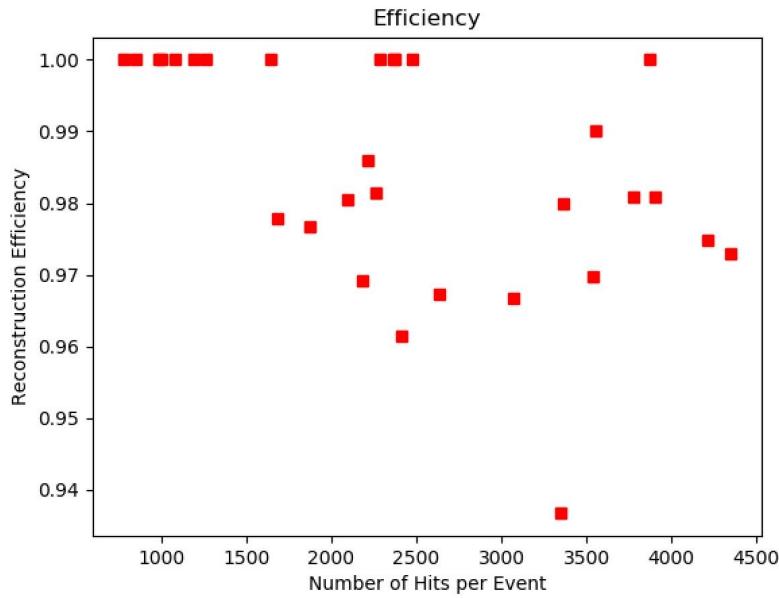


Figura 6.6: Gràfica de l'eficiència de l'algoritme de reconstrucció optimitzat en funció del nombre de *hits* per esdeveniment.

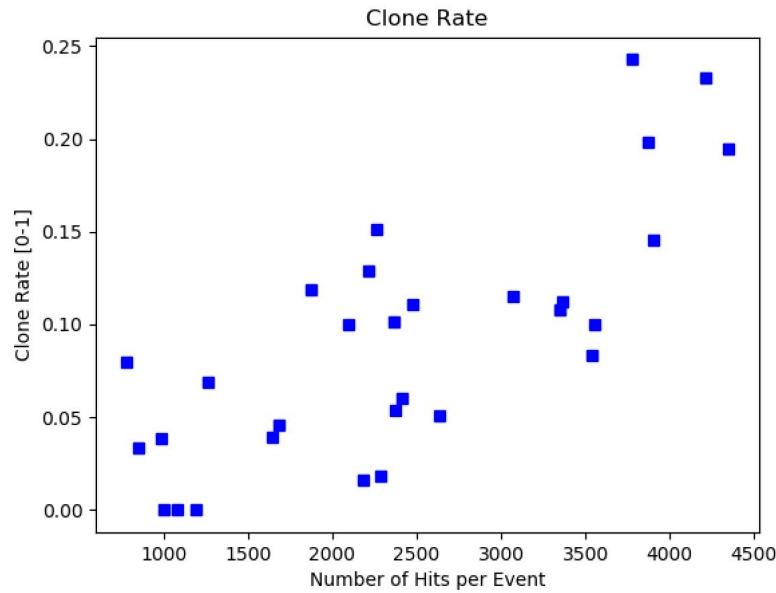


Figura 6.7: Gràfica de la taxa de *clone tracks* de l'algoritme de reconstrucció optimitzat en funció del nombre de *hits* per esdeveniment.

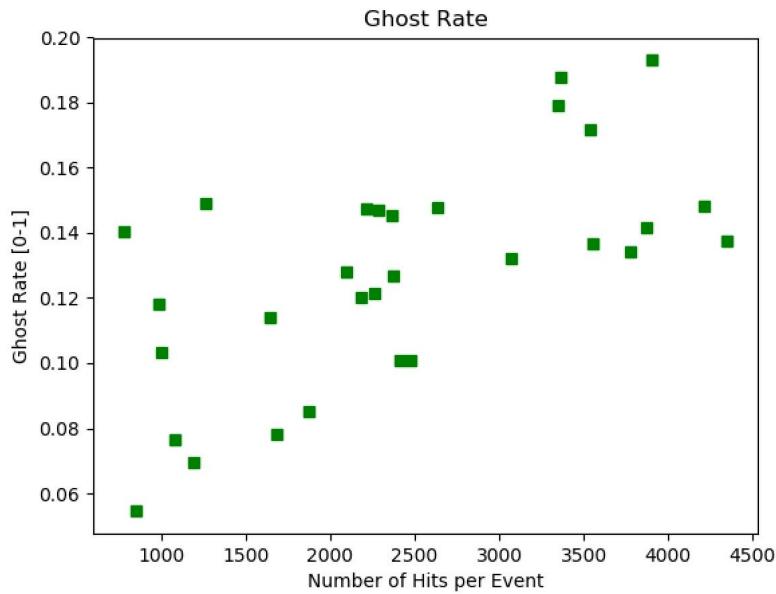


Figura 6.8: Gràfica de la taxa de *ghost tracks* de l'algoritme de reconstrucció optimitzat en funció del nombre de *hits* per esdeveniment.

Respecte a l'eficiència, com es pot veure a la figura 6.6, s'obtenen resultats molt bons, amb un valor mínim aproximat de 0.94. D'altra banda, tampoc presenta cap relació clara

amb el nombre de *hits* dels esdeveniments, tot i que els primers valors, aproximadament fins a 1500 *hits*, tenen la millor mitja d'eficiència de tots els esdeveniments.

Passant a la gràfica de la taxa de *clone tracks* que es mostra a la figura 6.7, presenta uns valors creixents amb el nombre de *hits* dels esdeveniments. També s'observa que pocs punts presenten una taxa de *clones* inferior a 0.05 i amb esdeveniments amb més de 3500 *hits* la taxa puja considerablement fins a un màxim de gairebé 0.25 sobre 1.

Finalment, respecte a la gràfica de la taxa de *ghost tracks* que es mostra a la figura 6.8, també s'observa un creixement, tot i que menys notable que en la gràfica anterior, a mesura que s'augmenta el nombre de *hits* en un esdeveniment. Tot i que els primers valors de la gràfica es dispersen en un marge de valors gran (entre 0.06 i 0.16), els valors finals presenten una taxa força elevada amb un màxim de gairebé 0.20 sobre 1.

6.7 Mesura de Temps

De la mateixa manera que s'ha mesurat el temps d'execució per l'algoritme sense les optimitzacions realitzades, s'ha repetit el procés, seguint la dinàmica de mesurar cent execucions pel mateix esdeveniment, i prenent com a resultat la mitjana per cadascun dels trenta esdeveniments disponibles. La figura 6.9 recull els valors de temps d'execució obtinguts en funció del nombre de *hits* de cada esdeveniment.

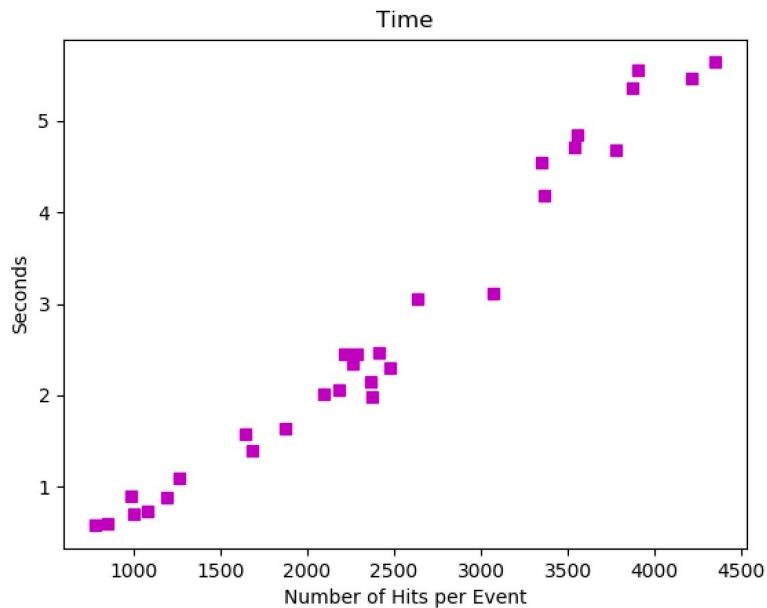


Figura 6.9: Gràfica del temps d'execució de l'algoritme optimitzat en funció del nombre de *hits* de cada esdeveniment.

Sobre la gràfica anterior es poden fer dues observacions importants, la primera és que

els valors de temps sobre els esdeveniments amb pocs *hits* són lleugerament majors que els obtinguts amb l'anterior algoritme, però la segona observació és que, en canvi, els temps d'execució amb esdeveniments amb major nombre de *hits* són molt inferiors als anteriors. Clarament, observant la gràfica de forma global, es pot veure que el creixement del temps d'execució amb el nombre de *hits* és lineal, gràcies a la utilització d'un arbre com a estructura principal en el cos de l'algorisme, amb un cost de cerca logarítmic que augmenta de forma lineal a mesura que s'insereixen més *hits* a l'arbre.

6.8 Optimització de Paràmetres

Per construcció, a l'algoritme implementat amb l'*R-Tree* hi ha dos paràmetres modificables que poden influir en el seu comportament i eficiència, aquests són, en primer lloc, el nombre de *hits* més propers a un donat (els seus veïns) per comprovar la seva compatibilitat per formar una traça. Com més veïns es retornin, més comprovacions s'hauran de fer, i per tant, el temps de computació augmentarà, però d'altra banda, si es redueix molt, l'eficiència de l'algoritme pot caure en no trobar aquells punts que formen una traça dins els veïns més propers retornats per l'arbre.

El segon paràmetre que pot ser modificat és la tolerància de la mesura de distància entre dos *hits* en el moment de decidir la compatibilitat per formar una traça. A mesura que s'augmenta la tolerància, es consideraran compatibles aquells punts lleugerament més separats, fet que pot comportar un augment de la taxa de *ghost tracks* tot i millorar l'eficiència. En canvi, si la tolerància es disminueix molt, pot fer caure l'eficiència de reconstrucció al no considerar compatibles *hits* que no estiguin estrictament a prop entre ells.

Amb l'objectiu d'optimitzar l'algoritme en funció dels dos paràmetres comentats, s'ha realitzat un estudi de l'eficiència, *clone rate*, *ghost rate* i temps d'execució, variant el nombre de veïns retornats en cada cerca a l'*R-Tree* implementat a l'algorisme des d'1 fins a 10 i també pels trenta esdeveniments disponibles per poder observar el comportament en funció del nombre de *hits* de cadascun. Per a fer-ho més fàcilment interpretable i visual, s'ha realitzat una gràfica de l'estudi del temps d'execució per separat a causa de la diferència d'escalatge. A la figura de l'annex A.1 es pot veure la gràfica en tres dimensions corresponent a l'estudi dels tres paràmetres d'eficiència en funció del nombre de veïns retornats en cada cerca i nombre de *hits* en cada esdeveniment. Per poder observar millor el comportament de l'algorisme en funció del nombre de veïns, es mostra la figura 6.10, corresponent a la mateixa gràfica tridimensional des de la perspectiva del nombre de veïns.

Respecte a la gràfica 6.10, l'eficiència és baixa en el cas d'utilitzar un únic veí, però a partir de dos veïns es manté a uns valors pròxims a 1 independentment que el nombre de veïns segueixi augmentant. Respecte a la taxa de *clones*, aquesta millora a mesura que augmenta el nombre de veïns, baixant fins a nivells que es mantenen pròxims a 0 independentment del nombre de *hits* per esdeveniment. En canvi, la taxa de *ghost tracks* augmenta de forma aproximadament lineal, com també ho fa el temps d'execució, tal com es pot veure a la gràfica de la figura 6.11.

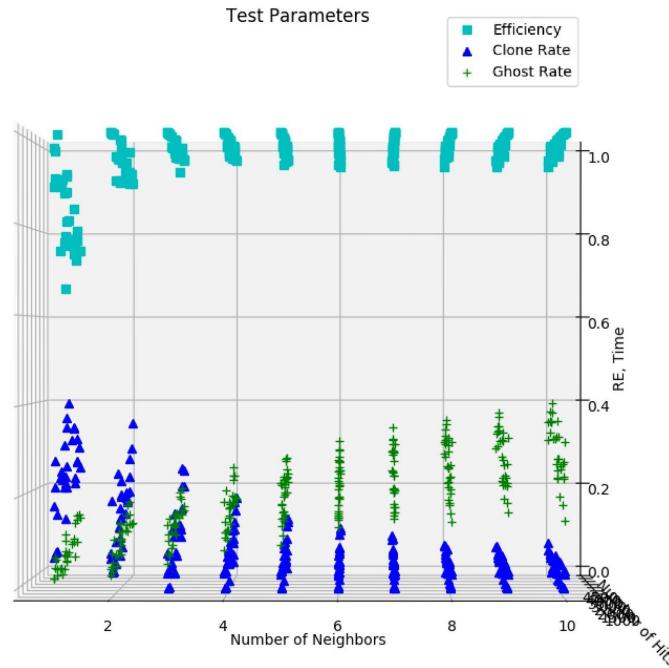


Figura 6.10: Gràfic tridimensional d'eficiència, *clone rate* i *ghost rate* en funció del nombre de veïns i el nombre de *hits* per esdeveniment, vist des de la perspectiva del nombre de veïns.

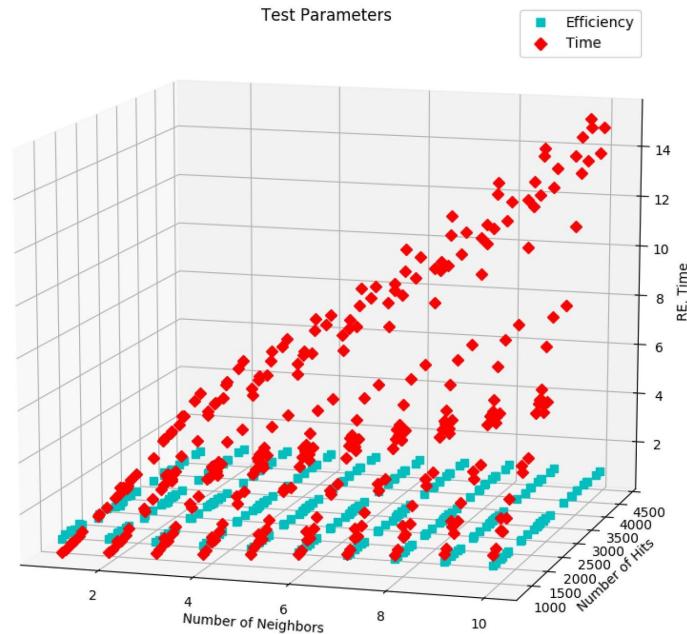


Figura 6.11: Gràfic tridimensional d'eficiència i temps d'execució en funció del nombre de veïns i el nombre de *hits* per esdeveniment.

Amb les gràfiques anteriors, es vol buscar el punt que maximitzi l'eficiència i a la vegada minimitzi la taxa de *clones* i de *ghosts* i el temps d'execució. Per fer-ho, s'han normalitzat tots els valors de temps mesurats per acotar-los entre 0 i 1, tal com estan evaluats els tres altres paràmetres. A continuació s'han sumat els paràmetres a maximitzar, invertint els que es volen minimitzar de la següent manera:

```
for i in range(len(RE)):
    normalized_parameters.append(RE[i] + (1-CR[i]) + (1-GR[i]) +
        (1-time[i]))
```

On *RE* guarda els valors d'eficiència de reconstrucció, *CR* fa referència a la taxa de *clone tracks*, *GR* fa referència a la taxa de *ghost tracks* i *time* fa referència a les mesures de temps d'execució. Amb els paràmetres normalitzats, la gràfica resultant en funció del nombre de veïns i el nombre de *hits* es pot veure a l'annex A.2, i projectant les dades sobre el nombre de veïns per observar millor els resultats s'obté la gràfica de la figura 6.12.

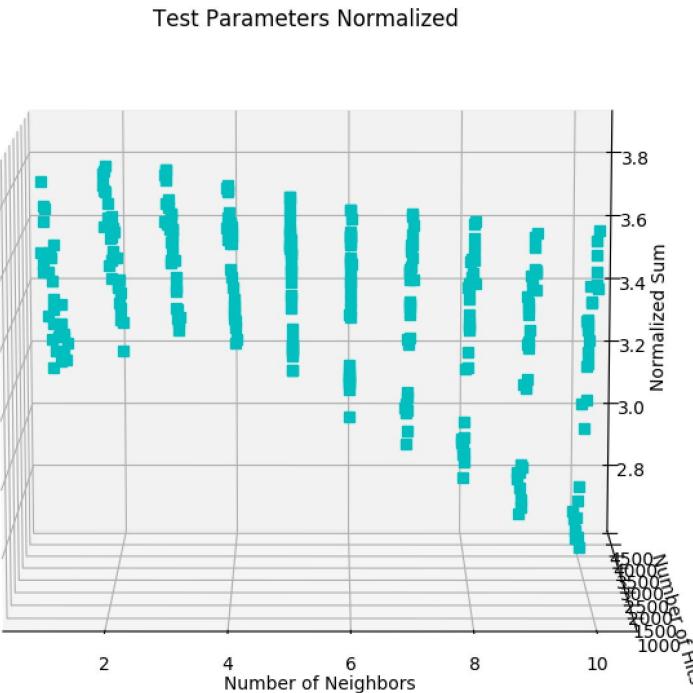


Figura 6.12: Gràfic tridimensional de la suma dels valors normalitzats d'eficiència, *clone rate*, *ghost rate* i temps d'execució en funció del nombre de veïns i el nombre de *hits* per esdeveniment, projectat sobre el nombre de veïns.

Com que els valors de la gràfica corresponen a la suma de quatre valors normalitzats [0 – 1], el valor del paràmetre corresponent al nombre de veïns que ens interessa és aquell

que pren els valors més pròxims a 4 i que a la vegada, decaiguin el mínim possible a mesura que augmenta el nombre de *hits* dels esdeveniments. A partir de la gràfica anterior, s'observa que els valors de nombre de veïns que millor compleixen les condicions exposades són 2, 3 i 4 veïns.

Amb aquests resultats, sense poder conoure un sol valor d'entre els tres millors amb les dades obtingudes fins al moment, es procedeix a avaluar el comportament de l'algoritme enfront de la variació de la tolerància pels tres valors de nombre de veïns que han donat els millors resultats.

La figura de l'annex A.3 mostra una gràfica tridimensional resultant de l'estudi del comportament de l'algoritme variant les toleràncies de 0.1 fins a 1.0 amb salts de 0.1 i en funció del nombre de *hits* dels esdeveniments, amb el paràmetre de veïns fix a 2. Per observar millor els resultats de la gràfica la figura 6.13 mostra la projecció de la gràfica A.3 sobre l'eix de tolerància, on es pot veure que d'eficiència comença a disminuir aproximadament a partir del valor de tolerància 0.4, el *ghost rate* presenta un creixement lineal a mesura que s'augmenta la tolerància i el *clone rate*, tot i mantenir-se força constant, té un creixement major en funció del nombre de *hits* per baixes toleràncies.

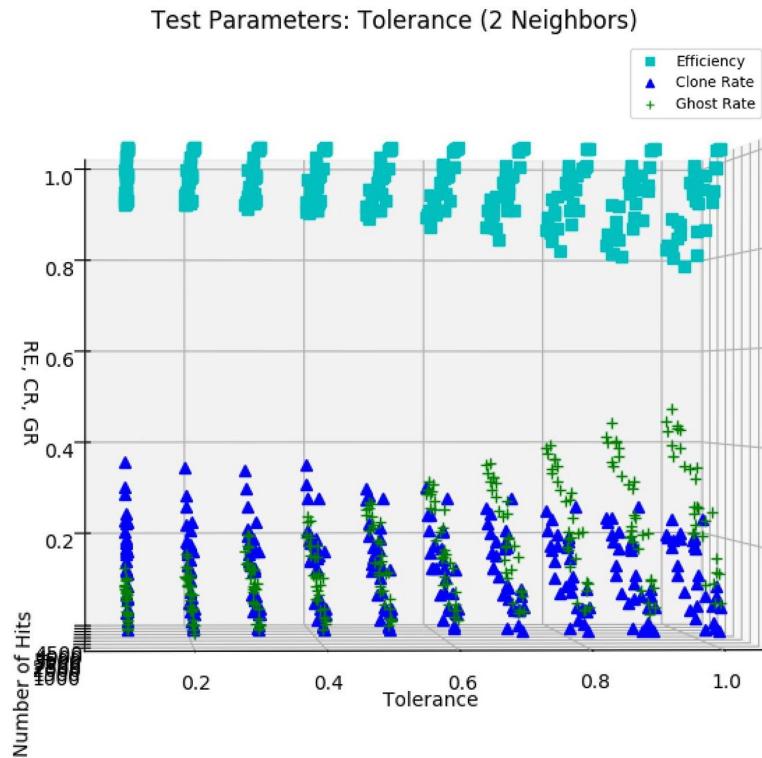


Figura 6.13: Gràfic tridimensional d'eficiència, *clone rate* i *ghost rate* en funció dels valors de tolerància i el nombre de *hits* per esdeveniment, vist sobre l'eix de tolerància.

Per no saturar les gràfiques d'eficiència, s'ha fet l'estudi referent al temps d'execució en

funció de la tolerància i nombre de *hits* per separat, tal com es pot veure a la gràfica de la figura 6.14. On s'observa que el temps d'execució no presenta una variació significativa enfront de la variació de tolerància, tot i tenir una millor resposta per a elevats nombres de *hits* per esdeveniment amb toleràncies més altes.

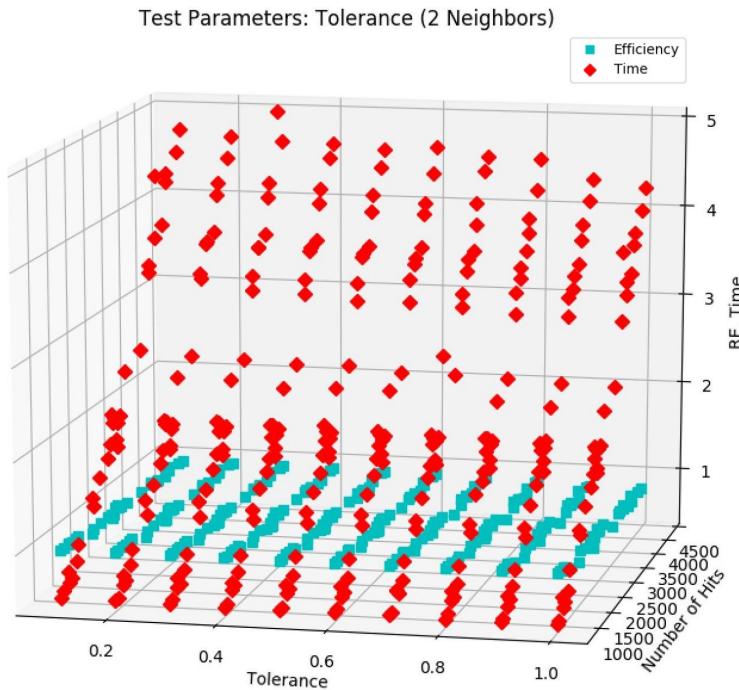


Figura 6.14: Gràfic tridimensional d'eficiència i temps d'execució en funció de la tolerància i el nombre de *hits* per esdeveniment, amb cerques de dos veïns.

A les figures de l'annex A.6 i A.7, es poden veure els mateixos estudis de temps realitzats pels paràmetres de cerca de valors 3 i 4 veïns.

A continuació s'ha repetit el mateix estudi anterior amb els dos millors valors restants del paràmetre de veïns, extrets del primer estudi, 3 i 4 veïns. Els resultats de les gràfiques obtingudes es troben a les figures A.4 i A.5 de l'annex. Per poder valorar els quatre paràmetres sota estudi (eficiència, *clone rate*, *ghost rate* i temps d'execució) de manera objectiva i poder escollir tant la tolerància més òptima com el nombre de veïns de cada cerca més òptim, s'ha generat una gràfica normalitzant els resultats obtinguts i sumant-los, de la mateixa manera que s'ha fet per l'estudi del nombre de veïns. Els resultats obtinguts de la gràfica en tres dimensions, amb les dades referents als tres possibles nombres de veïns superposades, es poden veure a la figura de l'annex A.8, i més en detall, amb les dades projectades sobre l'eix de tolerància a la figura 6.15.

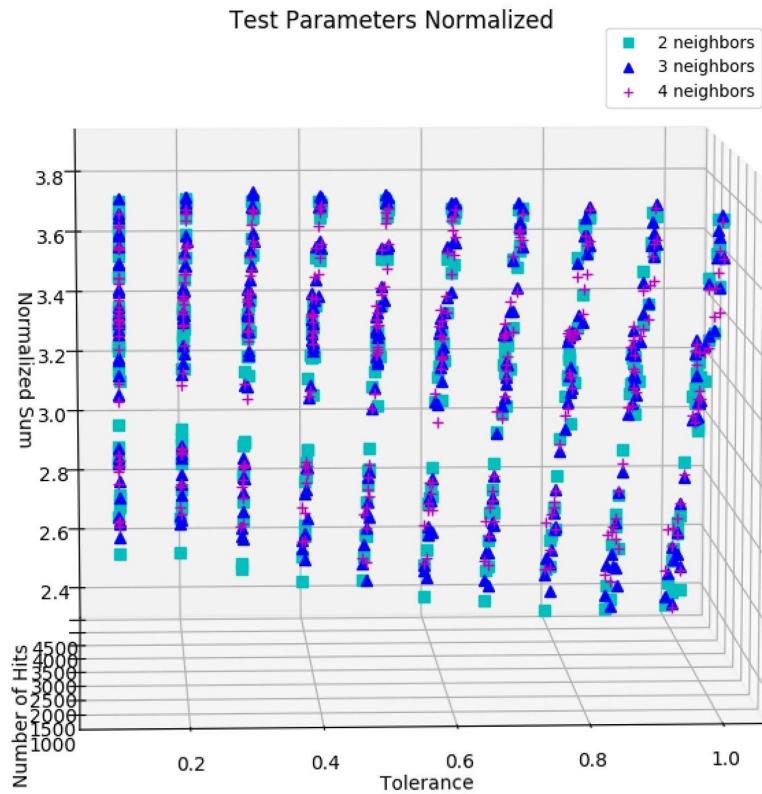


Figura 6.15: Gràfic tridimensional d'eficiència, *clone rate* i *ghost rate* en funció dels valors de tolerància i el nombre de *hits* per esdeveniment, vist sobre l'eix de tolerància i superposant els resultats de diferents nombres de veïns.

A la gràfica anterior es pot veure que la suma dels resultats normalitzats pren el valor més alt i a la vegada més compacte (que varia menys en funció del nombre de *hits*) amb el valor de tolerància 0.2. S'assigna doncs, aquest valor com la tolerància òptima. Pel que fa als tres nombres possibles de veïns extrets en cada cerca, s'ha realitzat un estudi de la distribució dels paràmetres normalitzats de cadascun, en funció del nombre de *hits* de cada esdeveniment.

La gràfica resultant es pot veure a la figura 6.16, on es pot veure com la mediana més alta de les tres distribucions és la corresponent a la cerca de quatre veïns, indicada amb una línia sobre la gràfica. De la mateixa manera, la mitjana més alta, indicada amb un triangle verd, és també la corresponent a la cerca de quatre veïns. Així doncs, tot i que la distribució pren el valor màxim en la cerca de tres veïns, es pren com a valor òptim el nombre de veïns de cada cerca a 4.

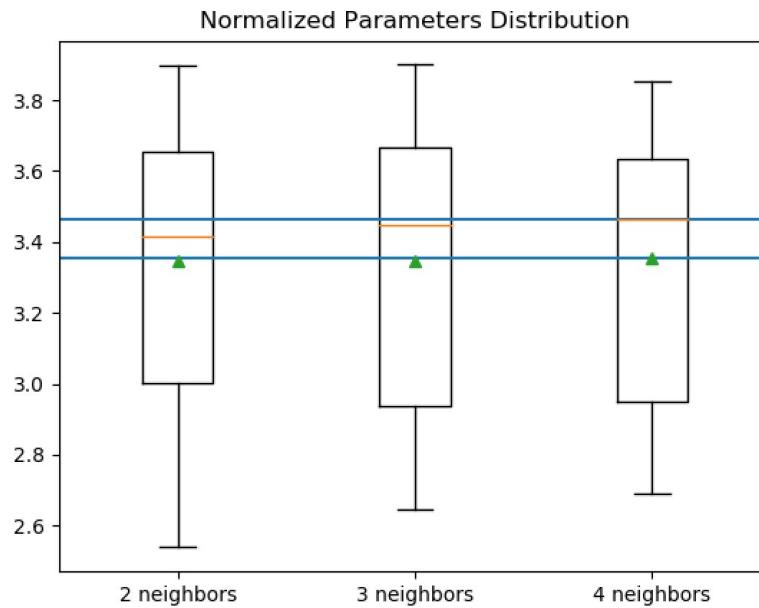


Figura 6.16: Diagrama de caixes de la variació de la suma de valors normalitzats en funció del nombre de *hits* de cada esdeveniment per a 2, 3 i 4 veïns.

6.9 Comparació de Resultats

Després d'estudiar el comportament dels dos algoritmes de reconstrucció de traces per separat, l'original, basat en l'autòmat cel·lular i implementat amb llistes, i l'optimitzat, amb la mateixa dinàmica però utilitzant un *R-Tree* com a estructura de dades i una taula de *Hash*, s'ha procedit a quantificar la diferència entre els paràmetres avaluats per tal de saber si s'ha aconseguit una millora significativa o, en cas contrari, quines són les especificacions de l'algoritme que cal millorar.

En primer lloc es compara l'eficiència de reconstrucció de traces. Com es pot veure a la figura 6.17, els valors d'eficiència obtinguts són similars en ambdós algoritmes en funció del nombre de *hits* de cada esdeveniment. Però per tenir una idea més clara de la diferència entre els dos algoritmes, s'ha fet un diagrama de caixes per veure la distribució de cadascuna de les dues eficiències i poder comparar-les més fàcilment.

Es pot veure al diagrama de la figura 6.18 que, tant la mitjana com la mediana d'eficiència de l'algorisme optimitzat amb l'*R-Tree*, són majors a les de l'algorisme original, fet que ens assegura el bon funcionament de l'algorisme optimitzat tot i treballar amb un nombre elevat de *hits* en cada esdeveniment.

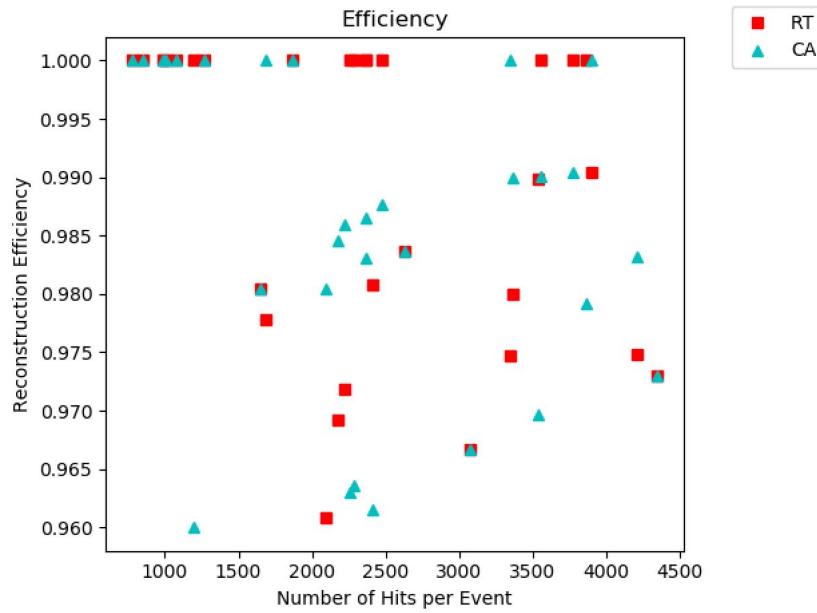


Figura 6.17: Gràfic comparatiu d'eficiència en funció del nombre de *hits* per esdeveniment entre l'algoritme original i l'optimitzat.

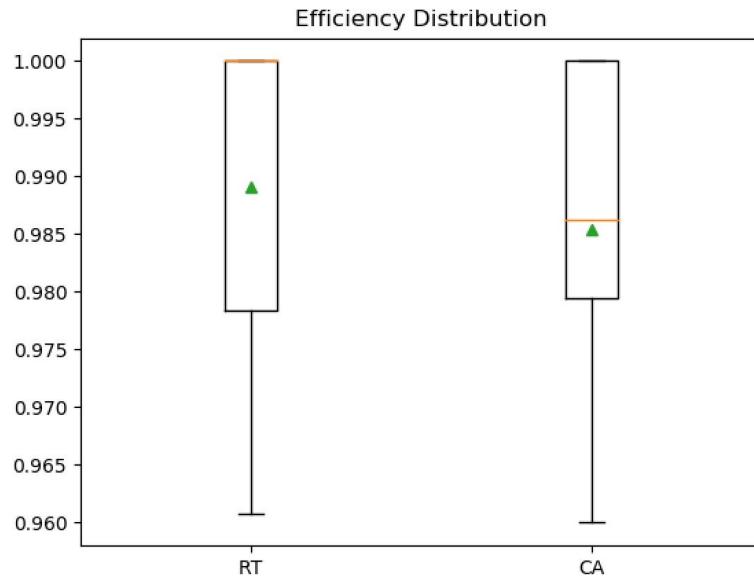


Figura 6.18: Diagrama de caixes de la variació d'eficiència en funció del nombre de *hits* per esdeveniment de l'algoritme original i l'optimitzat. La mitjana de la distribució es marca amb un triangle verd i la mediana amb una línia groga.

Cal tenir en compte que, tot i els bons resultats de l'eficiència, la taxa de *clone tracks* i la taxa de *ghost tracks* també són paràmetres molt importants a tenir en compte per l'efectivitat de l'algoritme. A la figura 6.19 es comparen els valors de *clone rate* d'ambdós algoritmes. Encara que per esdeveniments amb pocs *hits* els valors siguin similars, a mesura que s'augmenten, l'algorisme optimitzat presenta un creixement de la taxa de *clone tracks* més pronunciat que l'algoritme original.

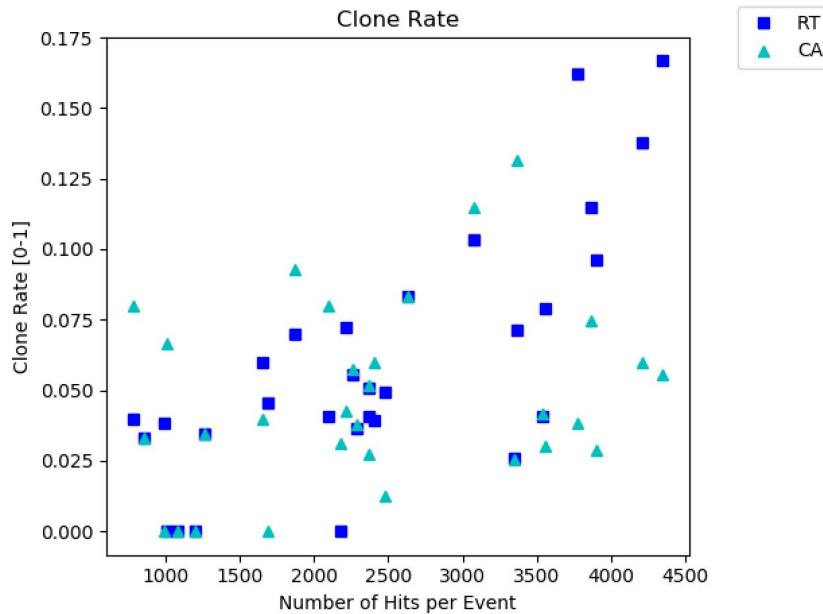


Figura 6.19: Gràfic comparatiu de *clone tracks* en funció del nombre de *hits* per esdeveniment entre l'algoritme original i l'optimitzat.

A la gràfica de l'annex A.9 es confirma el fet que tant la mitjana com la mediana de les distribucions de *clone tracks* de l'algoritme optimitzat són lleugerament majors que les de l'algoritme optimitzat, fet que perjudica l'efectivitat global per la reconstrucció de les traces dels esdeveniments.

Pel que fa a la taxa de *ghost tracks*, a la gràfica de la figura 6.20 es pot veure clarament que els valors obtinguts amb l'algoritme optimitzat són majors als de l'original amb un *offset* aproximat d'un 10%. També es pot apreciar aquesta clara diferència al diagrama de caixes de la figura de l'annex A.10.

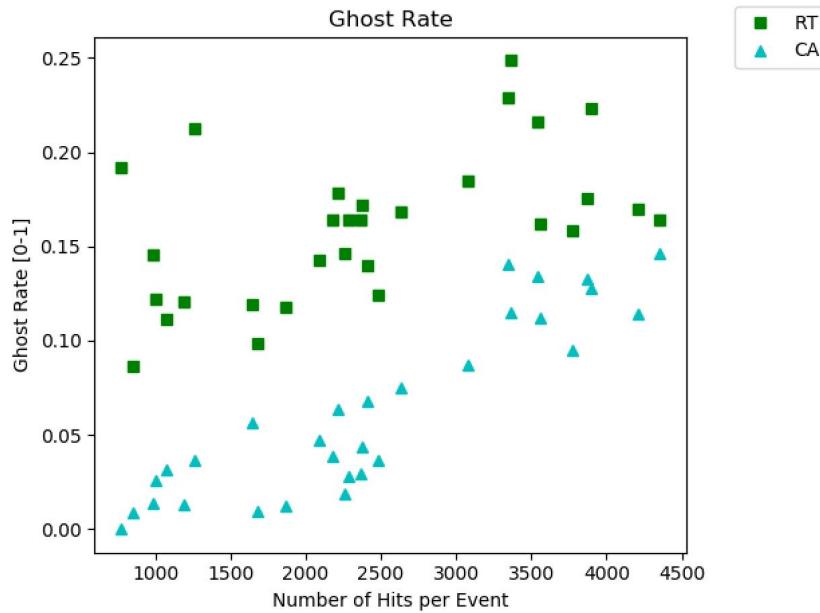


Figura 6.20: Gràfic comparatiu de *ghost tracks* en funció del nombre de *hits* per esdeveniment entre l'algoritme original i l'optimitzat.

Com a últim paràmetre, s'avalua la diferència entre els temps d'execució de cadascun dels algoritmes en funció del nombre de *hits* per esdeveniment. Com es pot veure a la figura 6.21, i com ja s'ha comentat a les respectives seccions (seccions 5.5 i 6.7) la corba que descriu el temps d'execució de l'algoritme original, a mesura que augmenta el nombre de *hits*, és exponencial. En canvi, amb l'algoritme optimitzat utilitzant l'arbre *R-Tree*, la corba de temps d'execució passa a ser lineal.

Cal tenir en compte que per a esdeveniments amb un baix nombre de *hits*, l'algoritme original presenta un temps menor, ja que el cost de construcció de l'arbre és fix per a totes les execucions i, en el cas de tenir pocs *hits*, és més eficient quant a temps fer una comparació per tots els *hits* adjacents que inserir-los tots a l'arbre i realitzar cerques sobre aquest. De tota manera, a mesura que s'augmenta el nombre de *hits* per esdeveniment i les llistes de l'algoritme original augmenten el seu nombre d'elements, el cost de cerca augmenta molt ràpidament, mentre que amb l'estructura de l'arbre, el cost de cerca és exactament el mateix amb independència del nombre de *hits* inserits, i per tant, l'augment de temps únicament és degut al fet que el creixent nombre de traces a reconstruir provoca més iteracions en el cos de l'algoritme.

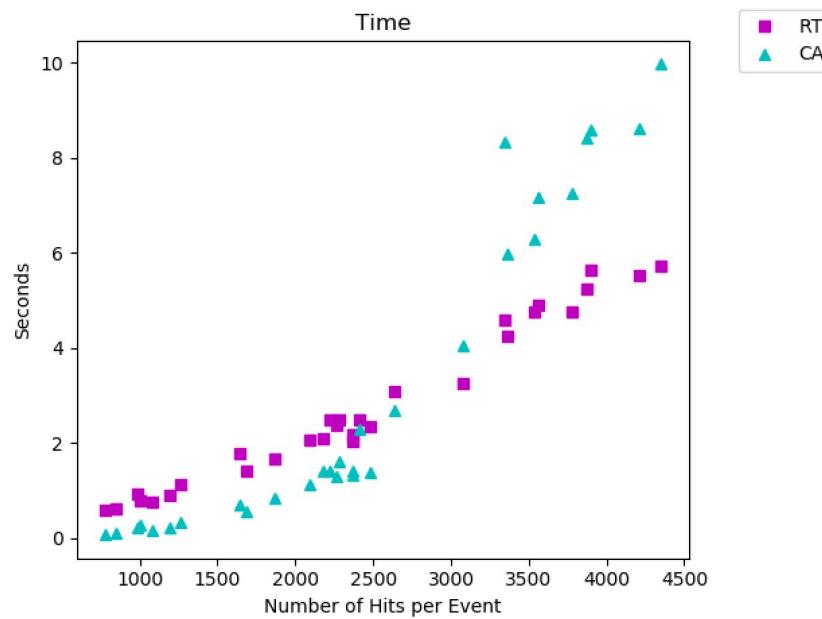


Figura 6.21: Gràfic comparatiu del temps d'execució en funció del nombre de *hits* per esdeveniment entre l'algoritme original i l'optimitzat.

Capítol 7

Conclusions

Com a resultats dels estudis realitzats sobre diferents algoritmes per la reconstrucció de traces del detector VELO d'LHCb, es pot concloure que no tots els algoritmes que funcionen correctament dins del seu àmbit, tenen la mateixa resposta aplicats a les dades del detector.

S'ha pogut comprovar amb l'estudi de la transformada de Hough que, encara que aquesta sigui una molt bona tècnica pel reconeixement de patrons sobre imatges digitals, pateix una saturació en aplicar-la sobre l'elevat i caòtic volum de dades provinents del VELO, que emmascara qualsevol informació interessant que es pugui extreure de la seva aplicació.

En canvi, amb l'adaptació del principi dels autòmats cel·lulars per la reconstrucció de traces, s'han aconseguit resultats coherents i amb una molt bona eficiència de reconstrucció. Tot i que, després de l'anàlisi a fons del comportament d'aquest algoritme envers el conjunt de dades del que es disposa, s'ha observat que la resposta del temps d'execució enfront del creixent nombre de dades a analitzar és exponencial. Aquest fet ha portat a qüestionar la implementació del cos de l'algoritme i s'ha trobat la necessitat de tenir present el tipus d'estructures de dades utilitzades amb l'objectiu de convertir el temps d'execució en una variable amb un creixement lineal.

D'altra banda, tenint present la imminent millora del detector per l'*upgrade* del pròxim LS2, es dona suport a la importància del temps d'execució emprat en la reconstrucció, ja que disposar d'un algoritme amb una resposta lineal presenta molts avantatges en un detector que espera augmentar encara més el nombre de dades adquirides en cada esdeveniment així com reduir encara més el temps de reconstrucció d'aquests.

Així doncs, s'ha optat per utilitzar, sobre el mateix algoritme, una estructura de dades de tipus arbre amb indexació espacial, l'*R-Tree*, amb el que s'ha aconseguit mantenir els resultats d'eficiència de reconstrucció similars als de l'algoritme previ i transformar la corba de temps d'execució en funció del nombre de dades a reconstruir d'exponencial a lineal.

A partir dels resultats obtinguts amb l'aplicació d'aquesta estructura de dades, es conclou la gran importància de tenir present el format de les dades que es disposen prèviament a la implementació de qualsevol algoritme, per així poder escollir amb criteri l'estructura més òptima i que pugui proporcionar el major nombre d'avantatges possibles. Concretament, el fet de treballar amb una estructura amb indexació espacial facilita en gran mesura la

implementació d'algoritmes de reconstrucció de dades, amb independència del seu propòsit final. Es manifesta doncs, la possibilitat de tenir present els arbres R de cara a futures implementacions per tècniques de reconstrucció, ja que, únicament per la seva resposta computacional, presenta un clar avantatge.

Anàlogament als resultats obtinguts amb l'algoritme de reconstrucció utilitzant R -*Trees*, es conclou que ha resultat molt interessant el fet de realitzar estudis en profunditat sobre les dades obtingudes com a resultat de la implementació, amb l'objectiu de quantificar i avaluar de forma numèrica i vertadera si els resultats obtinguts presenten realment una millora o no, ja que, en si mateixos, no donen més informació que la relativa a les execucions generades i subjectiva a la interpretació de les gràfiques generades. Gràcies a l'estudi d'optimitzacions de paràmetres, normalitzacions i ànalisi de distribucions, s'ha aconseguit donar un resultat objectiu de la millora i la significança d'aquesta.

Fora de l'abast d'aquest treball i com a continuació, es presenta la possibilitat d'adaptar la implementació d'alt nivell de l'arbre R -*Tree* de Python per poder estudiar més en profunditat l'estructura i la seva optimització variant constants que poden afectar el seu comportament, com per exemple el nombre de nodes o d'entrades en cada node. A més a més, cal estudiar fins a quin punt és possible parallelitzar l'algoritme implementat, ja que una de les línies de futur de l'experiment d'LHCb és la parallelització dels algoritmes per executar-los sobre GPUs amb l'objectiu de reduir el seu temps d'execució. És important destacar que l'algoritme implementat amb l' R -*Tree* presenta un cost computacional fix resultant de la inserció de totes les dades a l'arbre i present en totes les execucions sense excepció. El fet de parallelitzar aquesta operació podria reduir de forma constant el temps d'execució de les reconstruccions independentment del volum de les dades.

Altrament, cal estudiar amb més deteniment l'execució interna i el comportament de l'algoritme per esbrinar el motiu de l'increment considerable de la taxa de *ghost tracks* respecte a la versió original, ja que és un paràmetre important que condiciona la qualitat de la reconstrucció de forma directa.

Finalment, es vol destacar que, tant el CERN com l'LHCb són projectes immensament grans i que, en iniciar una col·laboració amb aquest grup de recerca, la magnitud i complexitat dels problemes que es presenten resulta aclaparador. A poc a poc, el procés inicial d'absorbir tota la informació possible ha anat convergint en un tema concret, com és la reconstrucció de traces, i en el que s'ha sabut aportar un punt de vista que no s'havia contemplat fins al moment i que resulta interessant, si més no, a tenir en compte, com és la utilització de diferents estructures de dades per l'optimització dels algoritmes.

Ha estat una gran experiència tenir l'oportunitat de conèixer el món de la recerca dins d'una institució com el CERN i m'agrada poder seguir col·laborant i aprenent cada vegada més amb aquest i tots els projectes que es plantegin.

Annex A

Gràfiques d'optimització de l'*R-Tree*

En aquest annex es troben totes les gràfiques extretes en els estudis sobre l'optimització dels paràmetres de l'algoritme implementat amb l'arbre *R-Tree*, i que complementen les explicacions del cos principal document.

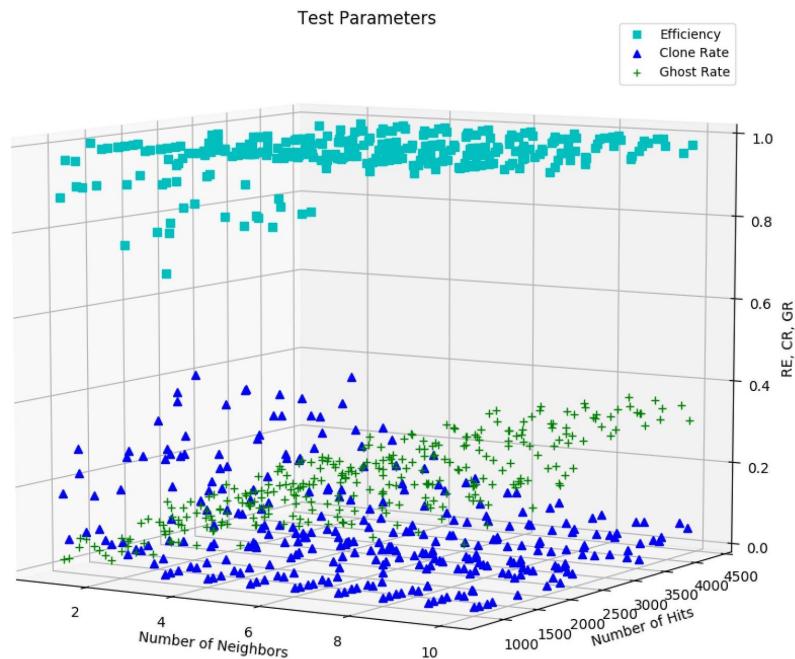


Figura A.1: Gràfic tridimensional d'eficiència, *clone rate* i *ghost rate* en funció del nombre de veïns i el nombre de *hits* per esdeveniment.

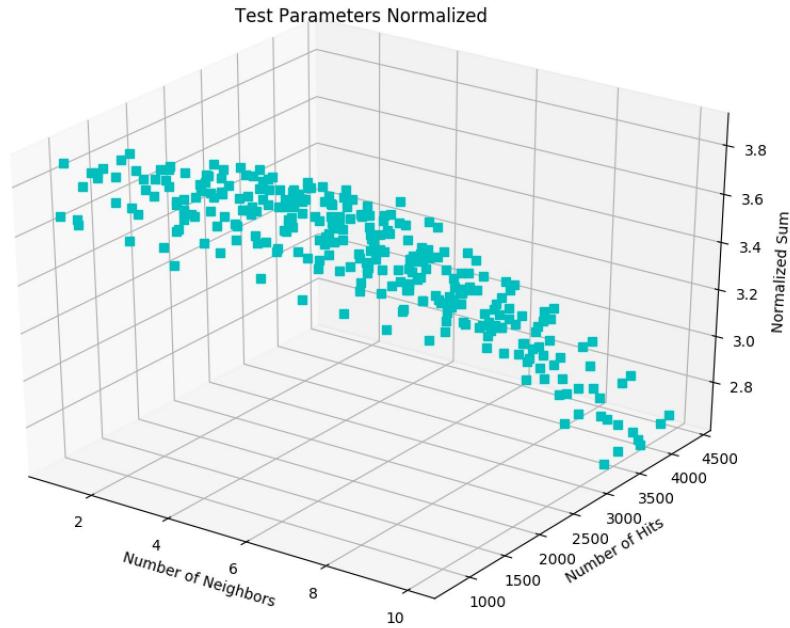


Figura A.2: Gràfic tridimensional de la suma dels valors normalitzats d'eficiència, *clone rate*, *ghost rate* i temps d'execució en funció del nombre de veïns i el nombre de *hits* per esdeveniment.

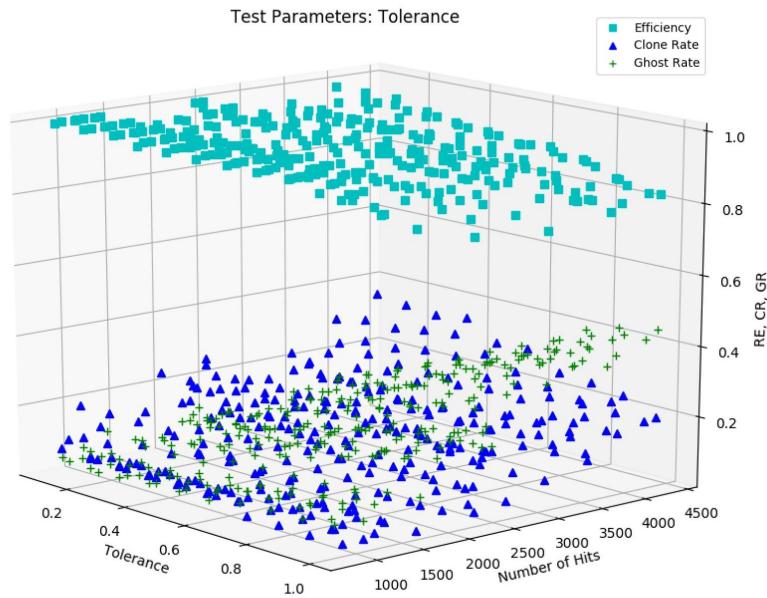


Figura A.3: Gràfic tridimensional d'eficiència, *clone rate* i *ghost rate* en funció de la tolerància i el nombre de *hits* per esdeveniment amb cerques de dos veïns.

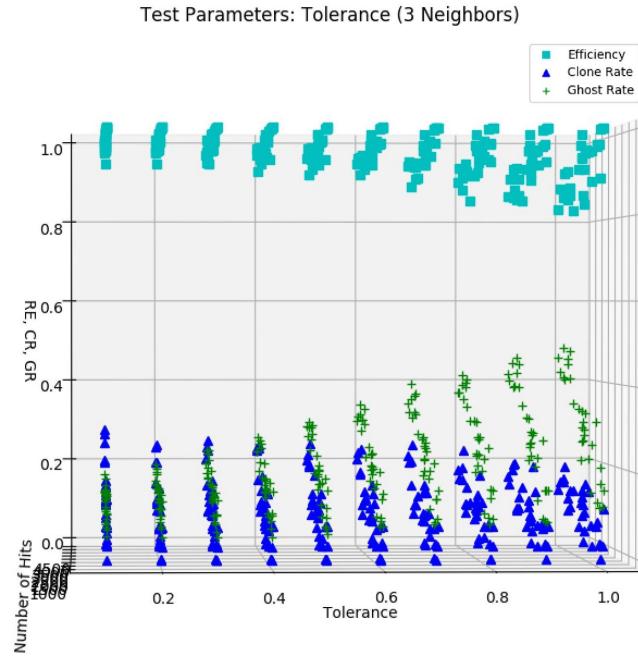


Figura A.4: Gràfic tridimensional d'eficiència, *clone rate* i *ghost rate* en funció de la tolerància i el nombre de *hits* amb cerques de tres veïns, sobre l'eix de tolerància.

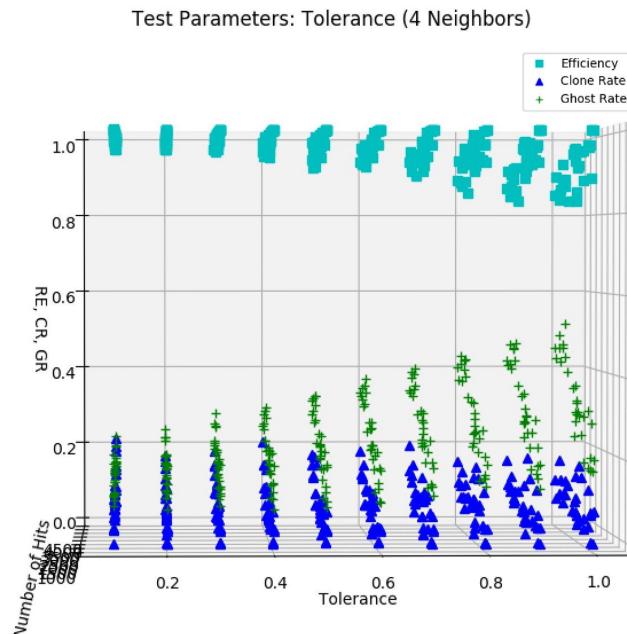


Figura A.5: Gràfic tridimensional d'eficiència, *clone rate* i *ghost rate* en funció de la tolerància i el nombre de *hits* amb cerques de quatre veïns, sobre l'eix de tolerància.

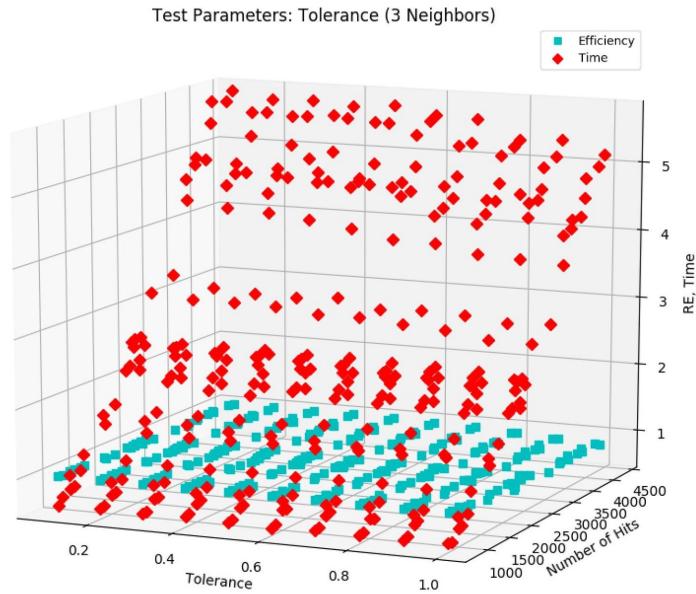


Figura A.6: Gràfic tridimensional d'eficiència i temps d'execució en funció de la tolerància i el nombre de *hits* per esdeveniment, amb cerques de tres veïns.

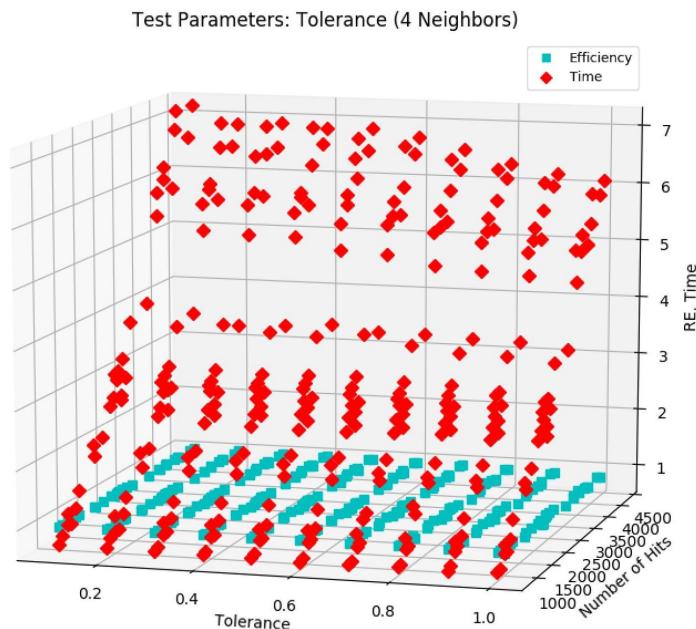


Figura A.7: Gràfic tridimensional d'eficiència i temps d'execució en funció de la tolerància i el nombre de *hits* per esdeveniment, amb cerques de quatre veïns.

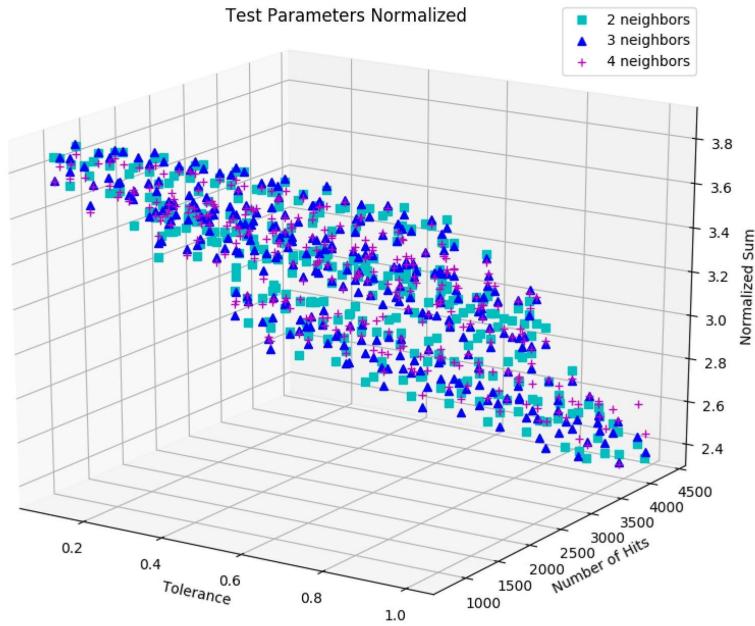


Figura A.8: Gràfic tridimensional de la suma dels valors normalitzats d'eficiència, *clone rate*, *ghost rate* i temps d'execució, referents als tres paràmetres de veïns diferents a estudiar, en funció de la tolerància i el nombre de *hits* per esdeveniment.

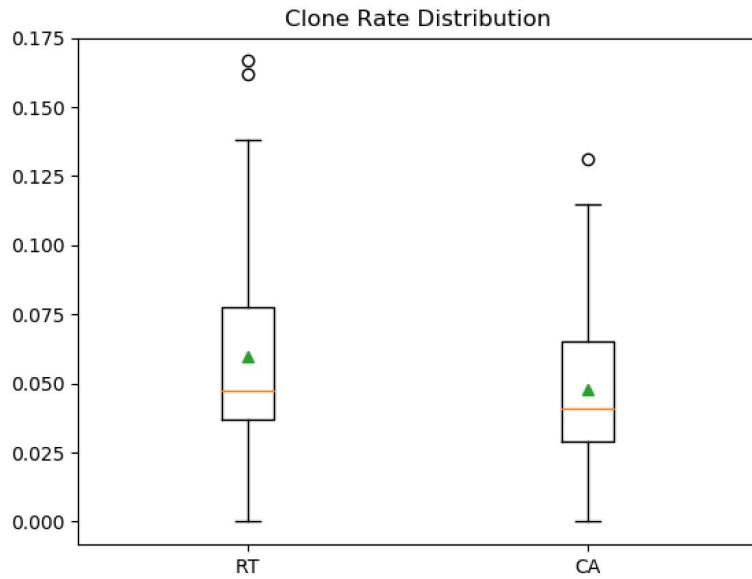


Figura A.9: Diagrama de caixes de la variació de *clone tracks* en funció del nombre de *hits* per esdeveniment de l'algoritme original i l'optimitzat. La mitjana de la distribució es marca amb un triangle verd i la mediana amb una línia groga.

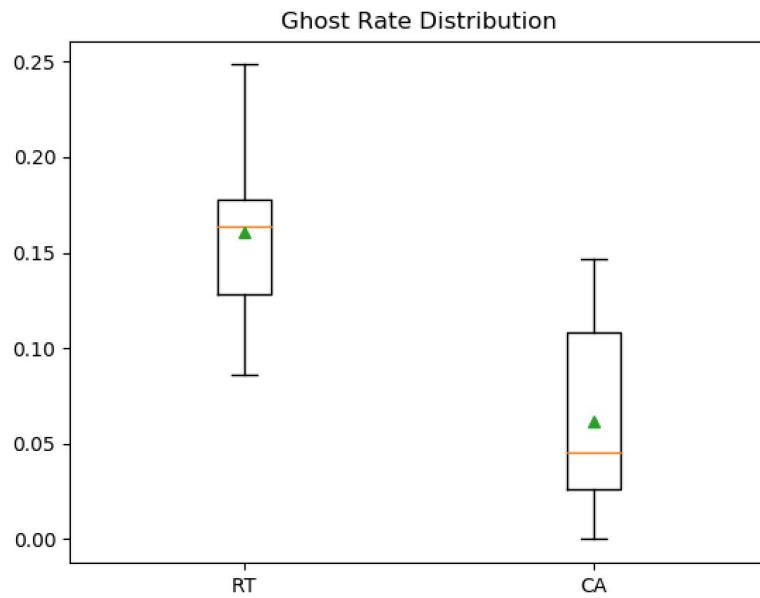


Figura A.10: Diagrama de caixes de la variació de *ghost tracks* en funció del nombre de *hits* per esdeveniment de l'algoritme original i l'optimitzat. La mitjana de la distribució es marca amb un triangle verd i la mediana amb una línia groga.

Bibliografia

- [1] S. Chatrchyan et al. “The CMS Experiment at the CERN LHC”. A: *JINST* 3 (2008), S08004. doi: 10.1088/1748-0221/3/08/S08004.
- [2] G. Aad et al. “The ATLAS Experiment at the CERN Large Hadron Collider”. A: *JINST* 3 (2008), S08003. doi: 10.1088/1748-0221/3/08/S08003.
- [3] K. Aamodt et al. “The ALICE experiment at the CERN LHC”. A: *JINST* 3 (2008), S08002. doi: 10.1088/1748-0221/3/08/S08002.
- [4] J Albrecht et al. *Implications of post-LS1 running conditions on LHCb’s data processing*. Inf. tèc. LHCb-PUB-2013-008. CERN-LHCb-PUB-2013-008. Geneva: CERN, juny de 2013. URL: <https://cds.cern.ch/record/1556085>.
- [5] A. Augusto Alves Jr. et al. “The LHCb Detector at the LHC”. A: *JINST* 3 (2008), S08005. doi: 10.1088/1748-0221/3/08/S08005.
- [6] A. Badalov et al. “LHCb GPU acceleration project”. A: *Journal of Instrumentation* 11.01 (2016), P01001. URL: <http://stacks.iop.org/1748-0221/11/i=01/a=P01001>.
- [7] T Bird et al. *VP Simulation and Track Reconstruction*. Inf. tèc. LHCb-PUB-2013-018. CERN-LHCb-PUB-2013-018. Geneva: CERN, oct. de 2013. URL: <https://cds.cern.ch/record/1620453>.
- [8] CERN. *About CERN*. 2015. URL: <https://home.cern/about>.
- [9] Geant4 Collaboration. *Geant4: A simulation toolkit*. Des. de 2017. URL: <http://geant4.web.cern.ch/>.
- [10] LHCb Collaboration. *LHCb Tracker Upgrade Technical Design Report*. Inf. tèc. CERN-LHCC-2014-001. LHCb-TDR-015. Febr. de 2014. URL: <https://cds.cern.ch/record/1647400>.
- [11] LHCb Collaboration. *LHCb VELO Upgrade Technical Design Report*. Inf. tèc. CERN-LHCC-2013-021. LHCb-TDR-013. Nov. de 2013. URL: <https://cds.cern.ch/record/1624070>.
- [12] CERN COURIER. *LHCb improves trigger in Run 2*. Set. de 2015. URL: <http://cerncourier.com/cws/article/cern/62495>.
- [13] Lyndon Evans i Philip Bryant. “LHC Machine”. A: *JINST* 3 (2008), S08001. doi: 10.1088/1748-0221/3/08/S08001.

- [14] Python Software Foundation. *The Python Standard Library*. 2018. URL: <https://docs.python.org/2/library/index.html>.
- [15] Sean Gilles Howard Butler Brent Pedersen et al. *Rtree: Spatial indexing for Python*. Des. de 2011. URL: <http://toblerity.org/rtree/index.html>.
- [16] Konrad Jende. *Measuring the D⁰ lifetime at the LHCb*. 2014. URL: <http://lhcb-public.web.cern.ch/lhcb-public/en/LHCb-outreach/masterclasses/en/DOLifetime.html>.
- [17] I. Kisel. “Event reconstruction in the CBM experiment”. A: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 566.1 (2006). TIME 2005, pag. 85-88. ISSN: 0168-9002. DOI: <https://doi.org/10.1016/j.nima.2006.05.040>. URL: <http://www.sciencedirect.com/science/article/pii/S0168900206008151>.
- [18] Mike Lamont. *LHC Report: Run 1 - the final flurry*. Febr. de 2013. URL: <https://home.cern/scientists/updates/2013/02/lhc-report-run-1-final-flurry>.
- [19] Ingo Lütkebohle. *Large Hadron Collider beauty experiment*. 2009. URL: <https://lhcb-public.web.cern.ch/lhcb-public/Welcome.html>.
- [20] John Von Neumann. *Theory of Self-Reproducing Automata*. Ed. de Arthur W. Burks. Champaign, IL, USA: University of Illinois Press, 1966.
- [21] Cian O’Luanaigh. *Seeing the invisible: Event displays in particle physics*. Juny de 2017. URL: <https://home.cern/about/updates/2015/06/seeing-invisible-event-displays-particle-physics>.
- [22] John Pavlus. *The Game of Life*. 2010. URL: <https://www.scientificamerican.com/article/the-game-of-life/>.
- [23] Daniel Hugo Cámpora Perez. *Tracking Wheels*. Private Communication.
- [24] Daniel Shiffman. *The Nature of Code: Simulating Natural Systems with Processing*. Ed. de Llc Theoklesia. Des. de 2012.
- [25] Torbjörn Sjöstrand. *Monte Carlo Event Generation for LHC**. Ag. de 1991. URL: <http://home.thep.lu.se/~torbjorn/preprints/th6275.pdf>.
- [26] T Szumlak i C Parkes. *Velo Event Model*. Inf. tèc. LHCb-2006-054. CERN-LHCb-2006-054. Geneva: CERN, oct. de 2006. URL: <http://cds.cern.ch/record/989093>.
- [27] Xabier Cid Vidal i Ramon Cid Manzano. *Taking a closer look at LHC*. 2015. URL: https://www.lhc-closer.es/taking_a_closer_look_at_lhc/1.home.
- [28] Eric Weisstein. *Cellular Automaton*. Maig de 2018. URL: <http://mathworld.wolfram.com/CellularAutomaton.html>.
- [29] Stephen Wolfram. *A new kind of science*. Ed. de Wolfram Media Inc. 2002.