MSE | MASTER OF SCIENCE IN ENGINEERING

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts Western Switzerland

Master of Science HES-SO in Engineering
Av. de Provence 6

CH-1007 Lausanne

Master of Science HES-SO in Engineering

Information and Communication Technologies (TIC)

Using a Quantum Annealer for particle tracking at LHC

Supported by the Lawrence Berkeley National Laboratory at Berkeley, CA (USA)

Under the supervision of Dr. Paolo Calafiura,
Chief software architect of the ATLAS experiment at CERN

Lucy Linder

Supervised by
Prof. Dr. Frédéric Bapst
At *HEIA-FR*

Expert
Noé Lutz, Engineering lead
Brain Applied Zurich – Google AI
At *Google*

Berkeley, HES-SO//Master, 2019

Information about this report

Contact information

Author:	Lucy Linder
	MSE Student
	HES-SO//Master
	Switzerland
Email:	lucy.linder@hefr.ch

Declaration of honor

I, undersigned, Lucy Linder, hereby declare that the work submitted is the result of a personal work. I certify that I have not resorted to plagiarism or other forms of fraud. All sources of information used and the author quotes were clearly mentioned.

Place, date: _____

Signature: _____

Validation

Accepted by the HES-SO//Master (Switzerland, Lausanne) on a proposal from:

Prof. Dr. Frédéric Bapst, Thesis project advisor

Noé Lutz, Google Zürich, Main expert

Place, date: _____

Prof. Dr. Frédéric Bapst	Philippe Joye
Advisor	Dean, HES-SO//Master

Acknowledgements

I would like to express my deep gratitude to Frédéric Bapst for his patient guidance, enthusiastic encouragement and useful critiques during our weekly Skypes. I am particularly grateful to Paolo Calafiura and Heather Gray for the amazing opportunity to do my thesis at LBL, for their trust, help and support throughout the project. My grateful thanks also go to Wim Lavrijsen, whose expert knowledge was of invaluable help. A special mention to Miha Muškinja for his idea of using impact parameters.

I would also like to extend my thanks to all the HEP.QPR team at LBL for their help as well as the technical and administrative staff who made my day-to-day activities at the lab easier.

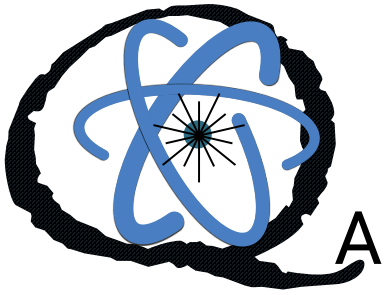
Abstract

The next upgrade of the LHC detectors will deliver far more data to be processed, which causes a strong pressure on the required computing power. The ATLAS team at LBNL is looking for new approaches that might help to boost the performance of their processing software libraries.

As part of this general effort, the present report discusses how to use Quantum Annealing in the context of particle tracking. A central pattern recognition task (known as track finding) is formulated in a way compatible with quantum annealers or other hardware dedicated to the Ising model. Our approach turns the classification task into a Quadratic Unconstrained Binary Optimization (QUBO) problem, where the binary variables correspond to triplets of 3D space points (hits), and the coefficients in the energy formula encode the expected physical properties of valid particle trajectories, using geometric features like curvature and other alignment constraints.

The model has been validated on datasets taken from a recent Kaggle challenge, and has been run through classical simulators as well as on commercial D-Wave machines. Our experiments tend to show that (i) the quality metrics (precision & recall) of the results meet the expectations, (ii) the model building time seems linear in its input (set of hits doublets), (iii) current D-Wave hardware yields results very similar to the classical solvers, and (iv) in the current setup it is more efficient to use the latter instead of a D-Wave, mainly due to an important communication/service overhead and the necessity to split large QUBOs into small instances that fit the hardware.

Keywords Quantum Annealing · Pattern Recognition · HEP Particle Tracking



Contents

1 Introduction

ATLAS is one of the four major experiments at the Large Hadron Collider (LHC) at CERN. The purpose of ATLAS is to test the predictions of the Standard Model of particle physics and to study the origins of the universe by boosting beams of protons to nearly the speed of light and observe the particles created by their collisions. The apparatus needed to witness those events is complex and includes many challenges, one of which being the reconstruction of the tracks followed by the charged particles as they fly through the detector. This task is known as *track finding* or *pattern recognition* and is the first step of a full *particle tracking* system. *ATLAS*

The High Luminosity upgrade (HL-LHC) expected for 2026 will increase the rate and the energy of the collisions, generating much more data to process. Current tracking algorithms scale badly with the number of particles ($> O(N^2)$). Latest projections based on expected funding and technology trends show that unless new tracking approaches are developed, there will be a shortage in CPU resources of a factor 3-10x [1]. This is why the ATLAS team is now exploring new approaches and unconventional hardware such as machine learning, dedicated FPGAs, neuromorphic chips and quantum computing. *HL-LHC*

This MSc thesis studies the applicability of a special instance of quantum computing, adiabatic quantum optimization, as a potential source of superlinear speedup for particle tracking. The goal is to develop a track finding algorithm that runs on a D-Wave machine and to discuss the interest of Quantum Annealing as opposed to more conventional approaches. *project goals*

This project is devoted to answer three central questions:

1. Can we apply quantum annealing (QA) to the problem of particle tracking ?
2. If so, is the chosen approach concretely executable on a real D-Wave ?
3. Is the chosen approach relevant and interesting to pursue further, and why ?

2 Track reconstruction

Chapter's content

2.1 Track reconstruction in HEP experiments

2.1.1 What is particle tracking

The LHC - *Large Hadron Collider* - experiment at CERN in Switzerland is the biggest particle accelerator ever built. It aims at discovering what our universe is made of by colliding particle beams at high speed, thus creating 'mini big bangs'. The result of those collisions is a firework of new particles, whose study can help understand what our universe is made of. *LHC*

To identify the type of and physical properties of those particles, complex *detectors* bend them using a strong external magnetic field and record the small energy they deposit (*hit*) when flying through different subdetector systems organized in layers. Without further analysis, it is not known which particle triggered which hits. *measurements*

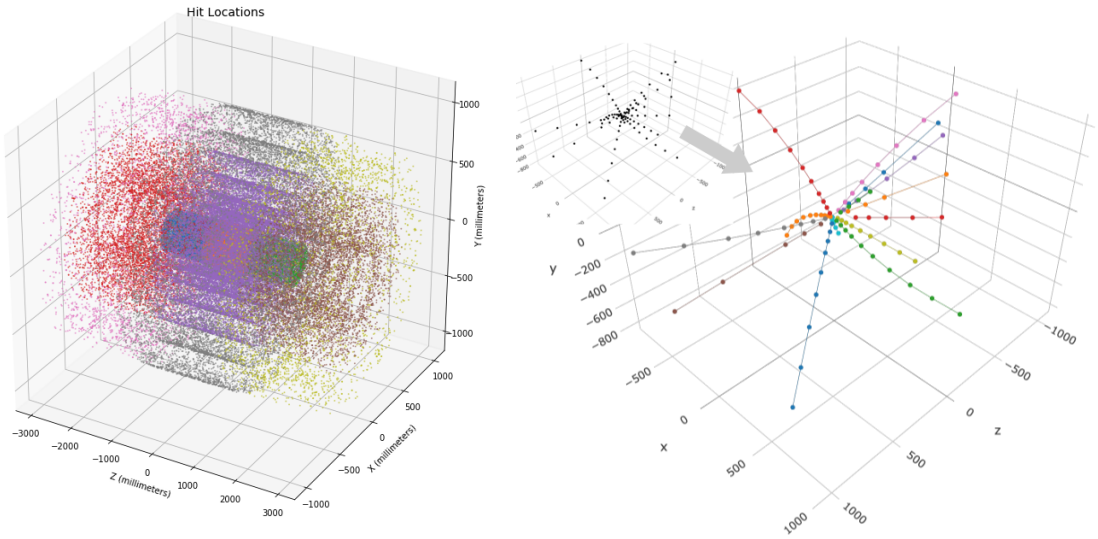
Particle tracking consists in reconstructing the trajectories of the charged particles from the detector's measurements (see Figure 2.1a). This task is usually divided into two steps: *particle tracking*

1. track finding or pattern recognition: this first step is a clustering task, where the goal is to assign hits to particles. The resulting clusters are called *track candidates*;
2. track fitting: for each cluster, apply a *fit*, i.e. determine the physical properties of the particles by following their trajectories.

This project focuses on step 1.

2.1.2 Overview and vocabulary

By convention, particles refer to the actual particles, i.e. the ground truth, while reconstructed particles paths are called tracks or track candidates. *particles / tracks*



(a) 3d cloud of hits from one TrackML event.

(b) Track reconstruction from a subset of hits.

Figure 2.1: Overview of particle tracking

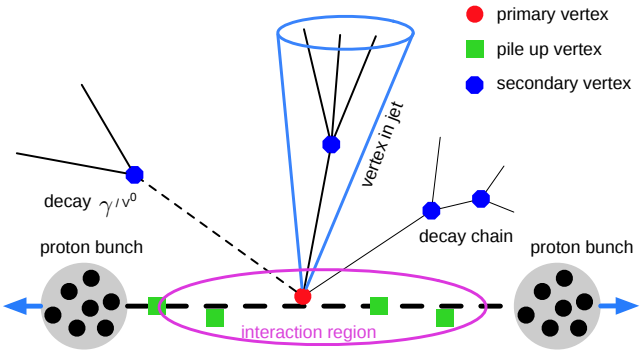


Figure 2.2: Kind of vertices

collisions

In the detector, collisions happen in a small longitudinal region near the center called the luminous region or interaction region. Collisions themselves are not recorded, only the particles that get created. The origin of one or more new particles is called a vertex. During a single beam-beam crossing, the average number of collisions mu (μ) is dictated by the density of the beams, or luminosity. At HL-LHC, the goal is to reach a luminosity of $\mu = 200$.

vertices

An event is the set of particle measurements (hits) in the detector associated to a single beam-beam crossing. Events are filtered online by so-called trigger systems, so that only "interesting" events are written to disk. The collision triggering the readout is called the primary vertex, while other collisions from the beams are called pile-up. Secondary vertices refer to other productions points, when particles are created from decay or hard-scattering. This is shown in Figure 2.2. Particles from the primary vertex usually have a high transverse momentum (p_t), making them especially interesting and easier to study.

spacepoints

At an abstract level, we can consider a hit as a spacepoint, that is a 3D measurement expressed in Euclidean coordinates. However, it is important to realize that spacepoints are reconstructed from more elementary signals and are thus not guaranteed to accurately reflect the actual particle coordinates. Consequently, they are only used in the pattern recognition, while the fitting programs usually rely on more granular information.

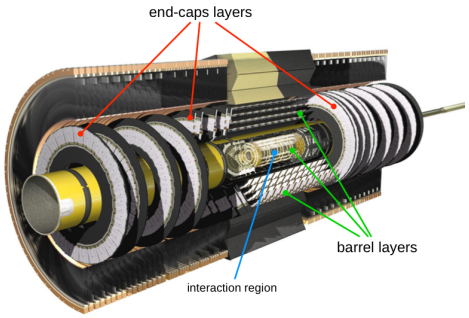


Figure 2.3: Barrel and end-caps layers in the ATLAS inner detector [2].

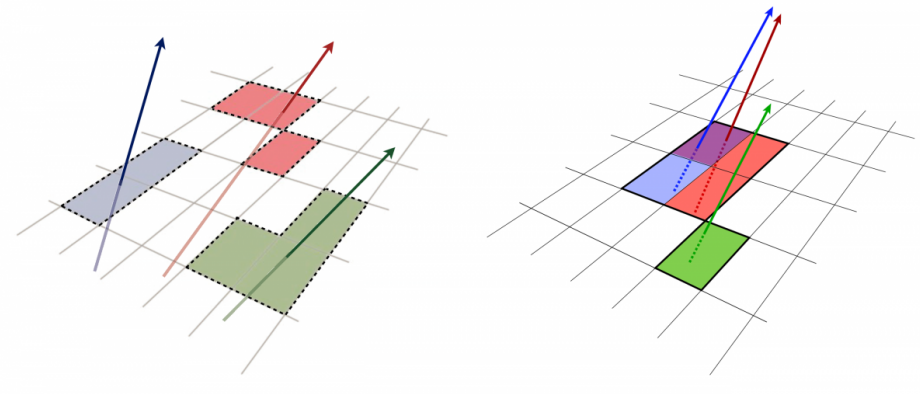


Figure 2.4: Illustration of isolated measurements (left) in the ATLAS pixel detector and merged measurements (right) due to very collimated tracks [3].

layers

As shown in Figure 2.3, detectors are made of discrete layers. Cylindrical-shaped layers surround the interaction region (this longitudinal section is referred to as the barrel) and wheel-shaped layers, called end-caps, are placed at each side of the barrel to provide a complete coverage. Usually, layers closer to the center of the detector offer the highest granularity and are arranged closer together. Each layer is made of a grid of sensing cells. The actual precision of the spacepoints depends on the technology in use (silicon pixels, silicon stripes, straws, etc.), the placement of the sensors and the readout channels.

imprecisions

The type and topology of the detector layers influence the measurements. In particular:

- spacepoints can only take discrete values;
- the more cells a particle activates, the less the precision of the reconstructed spacepoint (see Figure 2.4);
- particles can go through a layer undetected or a hit may not be recorded due to some thresholding (missing hits);
- one particle can leave multiple hits on the same layer (double hit) by activating multiple sensors;
- multiple particles can activate the same sensors, leaving only one hit;
- noise in the detector can trigger a readout, resulting in noise hits.

Coulomb scattering

Moreover, when particles go through a material such as a sensing cell, they can be disturbed (multiple Coulomb scattering) and change trajectory. This is especially true in the outer region of the detector, where the cells are thicker and made of other kinds of material.

2.2 Coordinate system

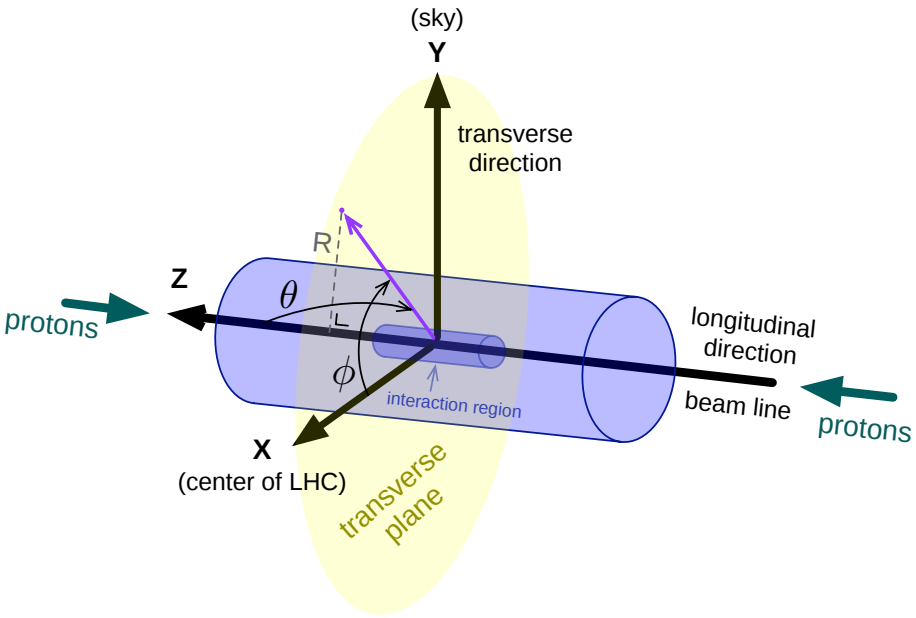


Figure 2.5: Coordinate system used in ATLAS

The coordinate system commonly used in ATLAS and other LHC experiments is presented in Figure 2.5. The interaction region is at the core of the detector and its center defines the origin of the coordinate system. The z-axis follows the beamline; the positive x-axis points to the center of the LHC ring; the positive y-axis points upwards to the sky (right-handed Euclidean coordinate system). The X-Y plane being perpendicular to the beamline, it is usually referred to as the transverse plane. *Euclidean*

Given the shape of the detector, cylindrical and polar coordinates are often used alongside the Euclidean coordinates. The azimuthal angle ϕ is measured around the beam from the x-axis. The polar angle θ is measured from the positive z-axis. The radial dimension, R, is the distance to the beamline, which is equivalent to the radius in the transverse plane. As such, the transverse plane is often described using $R - \phi$ coordinates. *cylindrical / polar*

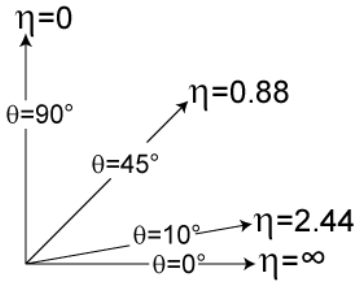


Figure 2.6: Pseudo-rapidity. The beamline is $\theta = 0$. Particles perpendicular to the beamline have $\eta = 0$, particles parallel to the beamline have $\eta \to \infty$ [4].

Another useful spatial coordinate in hadron collider physics is the pseudo-rapidity η (see Figure 2.6). It is given by: *eta*

$$\eta = -ln\Big[tan(\frac{\theta}{2})\Big]$$

η is often preferred over the polar angle θ because particle production is constant as a function of rapidity.

2.3 Particle trajectories

magnetic field

In order to measure the momenta and to contain the tracks of the particles, detectors are embedded into a strong magnetic field. Figure 2.7 shows the helicoidal trajectory followed by a charged particle with a constant velocity when moving through a magnetic field of strength B.

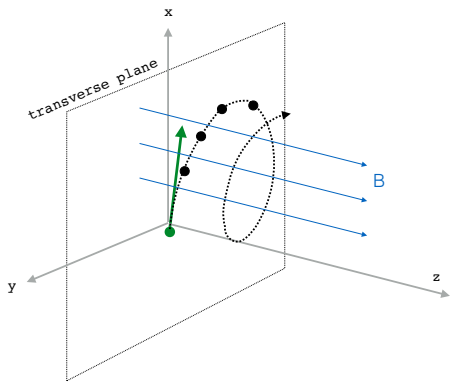
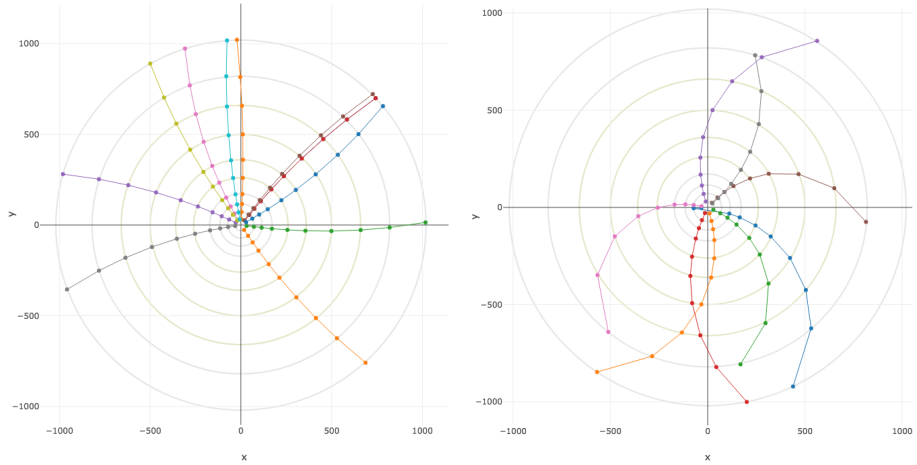


Figure 2.7: Illustration of a particle moving through a constant magnetic field [5].

X-Y plane

The magnetic field is aligned with the beamline (at least at the inner detector level), so that particles are bent in the transverse plane, drawing arcs of helices. Figure 2.8 shows the relationship between the transverse momentum (p_t) and the curvature: low p_t particles (i.e. parallel to the beamline) tend to bend a lot, while high p_t particles (i.e. perpendicular to the beamline) are almost straight. In real experiments, particles can depart from a true helix due to measurement errors, multiple Coulomb scattering and the fact that the magnetic field is not perfectly homogenous.



(a) High p_t particles.

(b) Low p_t particles.

Figure 2.8: Curvature of particles in the transverse plane depending on the transverse momentum (p_t): high p_t particles are almost straight.

R-Z plane

Most of the particles are created near the center of the detector and fly away from the collision point. As a result, the trajectories in the Z-R plane are usually straight, regardless of the momentum. This is shown in Figure 2.9. Exceptions are due once again to measurement errors and multiple Coulomb scattering.

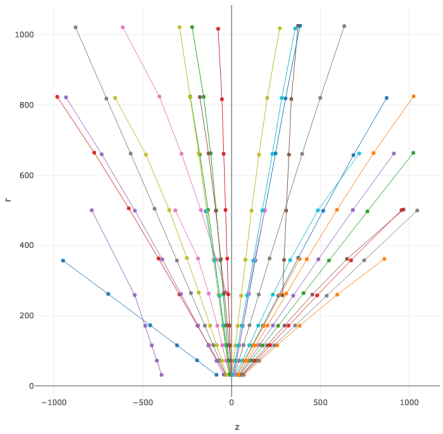


Figure 2.9: Particles usually follow straight lines in the Z-R plane.

2.4 The TrackML dataset

Kaggle challenge

As part of the general effort to find new ideas for track reconstruction, the ATLAS team ran a challenge on the Kaggle platform during the Summer 2018, called *TrackML*. The dataset provided for the challenge has been simulated using the realistic simulator ACTS [6] and is representative of real HEP experiments at high luminosity (mu (μ) > 140).

2.4.1 Dataset structure

files

The TrackML dataset is organized into events. Information about one event is split across four CSV files with the following naming convention: `event<eventID>-<filetype>.csv`. `eventID` is a nine digits long number left padded with zeros. `filetype` is one of:

- `hits`: hits information, including a `hit_id`, the Euclidean coordinates in millimeters (3D spacepoint) and the layer information;
- `cells`: information about the sensing cells used to reconstruct the spacepoints of each hit;
- `truth`: the ground truth for the event, linking hits to real particles;
- `particles`: parameter set for each particle, including the momentum and the number of hits.

test / train

The training dataset is composed of 100 events and includes all the files. The test datasets only include the `hit` and `cell` information, this is why they are not used in this study. Listing 1 shows the CSV header for each filetype. A more thorough description of each field is available in [7].

```
/* filetype: hits  */ hit_id,x,y,z,volume_id,layer_id,module_id
/* filetype: truth */ hit_id,particle_id,tx,ty,tz,tpx,tpy,tpz,weight
/* filetype: cells */ hit_id,ch0,ch1,value
/* filetype: particles */ particle_id,vx,vy,vz,px,py,pz,q,nhits
```

Listing 1: CSV header for each filetype.

event size

In the training dataset, the average number of particles per event is $10{,}792 \pm 1256$. A particle can leave between 1 and 20 hits, with an average of 12. The number of hits per event is $109{,}675 \pm 10{,}722$.

2.4.2 The TML detector

The TrackML measurements are based on an artificial detector called the TML detector.

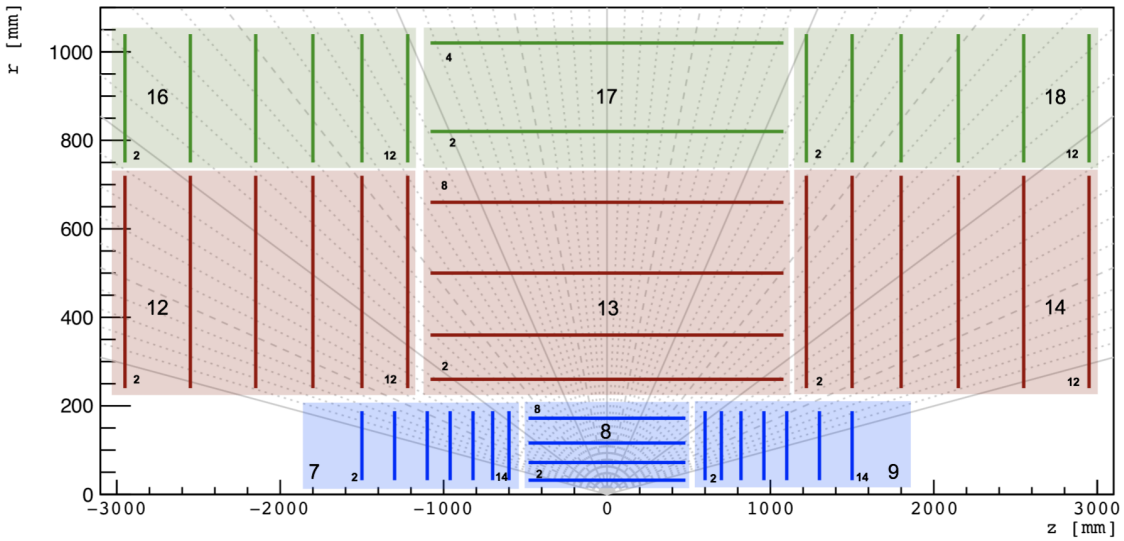


Figure 2.10: Longitudinal midsection plane of the TML detector showing the three volumes (in blue, red and green) along with their barrel and endcap layers [7].

topology

The TML detector is a simulation of a typical full Silicon LHC detector. It is made of three concentric devices (volumes) nested in one another, each of them composed of a cylindric structure in the barrel, and multiple discs on each side (end-caps). In the dataset, the barrels and end-caps have unique volume identifiers (`volume_id`). Each one is composed of several layers with an even `layer_id` unique within each volume. In total, the TML detector has 3 volumes and 10 layers. This is shown in Figure 2.10. A layer in the TML detector is built from slightly overlapping planar modules of sensitive devices (cells) that are mostly rectangular in shape (or trapezoidal for end-caps). They are identified using a `module_id`. This overlap is required to ensure hermetic coverage. Note that double hits may appear on the same layer, but never on the same module.

2.4.3 Dataset: general considerations

real conditions

The TrackML challenge task has been made almost as difficult as for real events. In particular:

1. each particle may leave multiple hits in a layer (double hits, $\sim 10\%$ of the time), or no hit at all (missing hits, $< 1\%$ of the time);
2. the simulated solenoid magnetic field has imperfections, causing distortions of the arcs of helices in the transverse plane;
3. multiple Coulomb scattering results in a stochastic deflection, meaning that a particle track can seem to change direction abruptly. The more material is traversed, the higher the probability of deflection;
4. the reconstructed positions (3D spacepoints) of the hits are derived from the cell information and offer only an approximation of the real (x, y, z) coordinates;
5. there is $\sim 15\%$ of noise hits per event; this is due to the inherent noise in the detector and the low MeV cut (150 MeV) used during the dataset generation.

simplifications

Beside the topology of the detector, there are also some differences with real events. The

most important for the task at hand is the fact that a hit can belong to at most one particle. This parameter is easy to control in simulation and simplifies the computation of the score, but does not reflect real HEP experiments.

On the other hand, some differences make the task more difficult. One example is the fact that the noise hits in the dataset do not have any correlation with material and physics signature.

2.4.4 TrackML score

one score

The Kaggle challenge is based on a single score, the TrackML score. It "is the intersection between the reconstructed tracks and the ground truth particles, normalized to one for each event, and averaged on the events of the test set" [8]. It is implemented in a helper library called `trackml-library` available on GitHub [8].

submission

To compute a score, one has to create a *submission*: for each hit in the dataset, assign one number to represent a particle id, using a random label for noise hits. This matches the output of a clustering task.

computation

The score computation procedure is thoroughly explained in [5], so we won't go into details here. But it is important to stress that:

- Each hit has a `weight` associated to it, depending on its position (points at each track extremity or close to the collision are more important) and the momentum of the track;
- Particles with three hits or less are considered spurious and thus don't count in the final score;
- Only valid associations count, meaning that assigning noise hits to particles is not penalized. In other words, the TrackML score measures the recall, but ignores the precision.

3 D-Wave

Chapter's content

This project focuses on quantum annealing, more specifically on the use of a D-Wave machine in the context of track reconstruction. D-Wave is a series of specialized quantum annealing hardware commercialized by D-Wave Systems. Their first model, released in 2011, is claimed to be "*the world's first commercially available quantum computer*" [9].

This chapter briefly introduces quantum annealing and the use of a D-Wave.

3.1 Basics of Quantum Annealing

adiabatic theorem

Quantum annealing (QA) relies on the adiabatic theorem of quantum mechanics [10] stating that a [perfectly isolated] physical system will remain in the ground state as long as:

(A) perturbations act slowly;
(B) there is a gap between the ground state and the rest of the system's energy spectrum.

QA recipe

Those two properties can be used to implement the equivalent of simulated annealing [11] in the following way:

1. start with a Hamiltonian H_0 whose ground state (as a quantum superposition) is easy to find and a problem Hamiltonian H_P whose ground state encodes the solution to a minimization problem;
2. define the combined Hamiltonian $H_{(S)} = A(s)H_0 + B(s)H_P$. $A(s)$ and $B(s)$ are functions that slowly decrease or increase, thus $H_{(s)}$ defines an interpolation from H_0 to H_P;
3. "run" the interpolation by introducing variations, making the system's ground state shift slowly from H_0's to H_P's ground state. The pace of the variation is given by $s = t/t_f \leq 1$, where t_f is called the *annealing time* and can be compared to the temperature in simulated annealing.

4. at the end of the interpolation, when $t = t_f$, the system's ground state encodes the solution to H_p.

If the annealing time is slow enough in relation to the gap between the two Hamiltonian's ground states, the final state of the system at $t = t_f$ encodes the solution.

3.2 QA in D-Wave

qubits

D-Wave computers are specialized quantum annealing hardware. They are made of qubits (quantum bits), circulating currents inside magnetic fields, that can represent 0, 1 or be in a superposition state (both 0 and 1). The set of qubits encodes the state of the system and cannot be set directly [12]. Instead, they can be influenced in two ways.

biases & couplers

First, one can apply a magnetic field in order to influence the probabilities of one qubit to collapse in one of the two states; the field strength is called a bias weight. Second, one can entangle qubits so that they tend to collapse either in the same or in opposite states; this is physically realized via couplers and controlled by a number called the coupling strength. Programming a D-Wave means specifying a set of bias weights and coupling strengths, which define together a Quantum Machine Instruction (QMI).

QUBO

QMIs can be expressed in two forms. In the Ising form, variables take values $\{-1, 1\}$, thus encoding a spin glass problem. In the QUBO form, variables take values $\{0, 1\}$, thus encoding a Quadratic Unconstrained Binary Optimisation (QUBO) problem. This study uses the latter and expresses objective functions/QMIs as shown in Equation 3.1. The influence of the linear and quadratic terms is summarized in Table 3.1.

$$O(a; b; q) = \sum_{i=1}^{N} a_i q_i + \sum_{i}^{N} \sum_{j}^{N} b_{ij} q_i q_j \quad q \in \{0, 1\} \tag{3.1}$$

bias weight a_i	influences one qubit q_i	-1	q_i tends to collapse into 0.
		1	q_i tends to collapse into 1.
coupling strength b_{ij}	influences two qubits q_i and q_j	-2	both q_i and q_j tend to collapse into 1.
		2	at least one of q_i and q_j tends to collapse into 0.

Table 3.1: Bounds and effects of the different weights used in a D-Wave QMI (QUBO form).

anneal

How does the D-Wave find the global minimum of a QMI ? Following the Quantum annealing recipe, the system starts in a ground state where all qubits are in superposition and there is no entanglement. Then, the problem Hamiltonian (QMI) is slowly introduced by activating the fields and couplers and lowering the influence of the initial Hamiltonian. This variation changes the energy landscape: from a single valley, minima begin to appear and barriers are raised. Qubits are still able to transition between states due to quantum tunneling[1], until the barriers become too high and the qubits "freeze" into their final state. Ideally, the whole process is adiabatic and the system stays in the ground state all along. In this case, the state in which the system collapsed at the end of the anneal is a global minimum. In current D-Wave's implementation, the default anneal time t_f is 20 µs.

[1]Transition can occur both by quantum-mechanical tunnelling and by absorbing heat (thermal excitation). This is why D-Wave processors are maintained at a temperature below about 45 mK: thermal excitation destroys the quantum nature of the qubit, and so must be avoided during quantum annealing [13].

limitations

While this sounds perfect in theory, certain factors may cause the system to jump from the ground state into a higher energy state. The first kind is linked to the fact that the D-Wave is a physical system and, as such, cannot be perfectly isolated from the outside world. Hence, thermal fluctuations and other external factors may interfere with the process. The second kind is more closely related to the problem at hand. For example, depending on the energy landscape of the problem Hamiltonian, the gap between the ground state and the first excited state might be too low to respect the adiabatic theorem.

low-energy solutions

Even though the probabilities of staying in the ground state can be small, the solutions returned by the D-Wave are usually of low energy and thus very helpful. However, due to noise and quantum probabilities, the anneal needs to be run multiple times and the different answers compared in order to get a exploitable result.

3.3 D-Wave hardware

number of qubits

Since 2011, D-Wave Systems released four generations of quantum annealing, increasing the number of qubits available. Starting at 128 qubits with the D-Wave One, the latest model has as much as 2048 qubits.

number of couplers

Inside the D-Wave, however, the qubits are not fully connected. Instead, they are arranged in a sparse graph with a structure known as a Chimera graph. As shown in Figure 3.1, this reduces significantly the number of couplers.

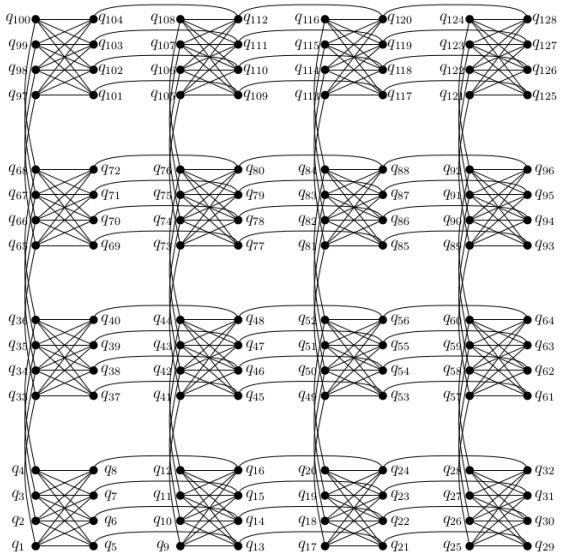


Figure 3.1: C16 chimera graph of the D-Wave 2000Q: 2048 qubits, 6016 couplers.

minor embedding

Due to the limited number of couplers, QUBOs with a high number of pair-wise connections have to be transformed in order to fit the hardware. This *mapping* process is called minor embedding and works by using multiple physical qubits, called chains, in order to represent one variable from the original QUBO and by adapting the weights accordingly. A complete explanation of how minor embedding work is available on the D-Wave documentation website[2] [16].

embedding tools

Finding the minor of a graph is an NP-hard problem [17]. Nonetheless, tools based on clever heuristics exist and one is directly integrated in the SAPI client libraries from D-Wave Systems. Called minorminer, it is also available as a standalone library on GitHub [18].

[2] See section "Minor-Embedding a Problem onto the Chimera Graph".

D-Wave's Chimera and CN notation

A Chimera architecture is composed of M unit cells of $2 \times L$ qubits organized as bipartite. Qubits on the left of the cell are connected (i.e. share a coupler with) to the vertical neighboring cell, while qubits on the right are connected to the horizontal neighbor.

The whole Chimera can be described using the CN notation: a CN Chimera is a Chimera of $N \times N$ cells of $N/2$ qubits. The D-Wave 2000Q, for example, uses a $C16$ chimera: its 2048 qubits are mapped into a 16×16 matrix of unit cells of 8 qubits [14], giving a total of 6016 couplers. This is shown in Figure 3.1.

Note that D-Wave Systems is currently working on a new architecture called *Pegasus* that will allow a qubit to couple with 15 other qubits instead of 6 [15].

3.4 Available D-Waves

two machines

D-Wave machines are highly specialized hardware. The initial and running costs are high, making them very scarce around the globe. For this project, we are lucky to have access to two machines: a D-Wave 2000Q available through the D-Wave Leap Cloud platform and a D-Wave 2X standing at the Los Alamos National Laboratory (LANL). Table 3.2 shows the specifications for both models.

Model	Release date	#qubits	#couplers	Location
D-Wave 2X	August 2015	1152	3360	LANL
D-Wave 2000Q	January 2017	2048	6016	Leap (cloud)

Table 3.2: Specifications of the two D-Wave models used in this project.

quota

Launched in October 2018, Leap [19] is a free service from D-Wave Systems offering 1 min of Quantum Processing Unit (QPU) time upon sign in. LANL is an LBL partner and gave us unlimited access to their equipment.

3.5 D-Wave Software Suite

Ocean software is a suite of tools from D-Wave Systems for solving hard problems with quantum computers. The suite is open source and available on the D-Wave GitHub repository [18]. This section gives an overview of two of its components. For more information, visit the Ocean Documentation [20].

3.5.1 SAPI

definition

SAPI (Solver API) is an application layer built to provide resource discovery, permissions, and scheduling for quantum annealing resources at D-Wave [3]. Both Leap and LANL D-Wave machines offer a SAPI endpoint, meaning that one can use the SAPI client to connect and submit problems in the same manner.

[3] See SAPI on Ocean's doc.

overview

A SAPI client only needs an initial endpoint and a token. It then uses HTTPS requests (REST) to communicate with backend servers. This includes the quantum server that sends problems to and returns results from the QPU, and post-processing servers that optionally process and transform the raw results. The SAPI libraries also define common abstractions and class hierarchies, so that classical and quantum samplers can be used interchangeably. Listing 2 shows a very simple example of creating and submitting a QUBO using a SAPI client in Python 3.

```
# import ocean tools
from dwave.system.samplers import DWaveSampler
from dwave.system.composites import EmbeddingComposite
# define a QUBO
Q = {(0, 0): 1, (1, 1): 1, (0, 1): 1}
# connect to a D-Wave using a config file...
dwave_sampler = DWaveSampler(config_file='/path/to/dwave.conf')
# ... or by giving the information directly
dwave_sampler = DWaveSampler(endpoint='https://endpoint.url', token='MY-TOKEN')
# use a minor-embedding wrapper around the solver, so that
# QUBOs will be embedded/un-embedded automatically
sampler = EmbeddingComposite(dwave_sampler)
# submit the QUBO to the D-Wave
response = sampler.sample_qubo(Q)
# get the best answer, along with its energy
best_energy = response.record.energy[0]
best_answer = next(response.samples())
```

Listing 2: Example on how to use the Ocean tools to submit a QUBO to a D-Wave in Python 3.

samplers

D-Wave Systems offers many types of samplers, from a quantum sampler to various simulation or classical solvers. All samplers inherit from the `dimod.Sampler` class, which defines three methods: `sample`, `sample_qubo` and `sample_ising`. Under the hood, both QUBO and Ising formulations are translated into a BQM – `BinaryQuadraticModel` – a common abstraction which encapsulates linear (bias weights) and quadratic weights (coupling strengths). This BQM is then fed to the `sample` function and the results are then converted back to their initial representation.

responses

The result of a `sample` call is standardized across samplers (class `dimod.Response`) and contains by default the list of unique solutions, along with their energies and number of occurrences, sorted by ascending energies. A solution contains the list of all variables (i.e. qubits) along with their final state ($\in \{0, 1\}$ for QUBO problems and is $\in \{-1, 1\}$ for Ising problems). It is also possible to ask the sampler to return a raw list of results. In this case, the response is simply the list of solutions in the order they were found along with their energy.

D-Wave sampler

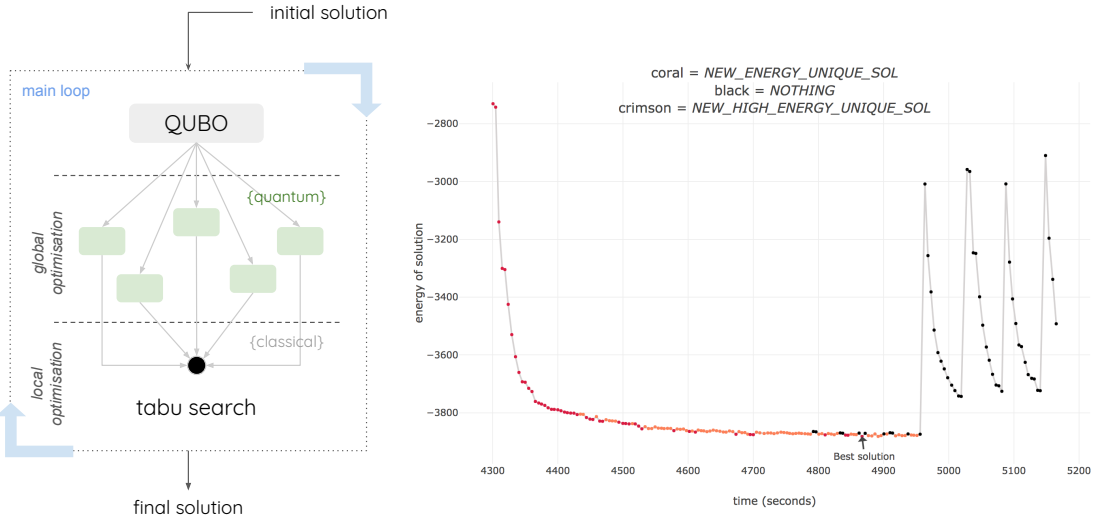
The `DWaveSampler` is the abstraction used to connect to a remote D-Wave using the SAPI interface. It comes with a lot of parameterization options, the most important one being the number of reads (`num_reads`, default to 50), that is how many times the anneal is performed. It doesn't handle minor embedding out-of-the-box, but can be wrapped in an `EmbeddingComposite` component that will automatically embed a BQM before sampling (and vice-versa).

3.5.2 qbsolv

overview

qbsolv [21] is a "*tool that solves large QUBO problems by partitioning into subproblems targeted for execution on a D-Wave system*" [22]. Compliant with the SAPI `Sampler` interface, it handles large and/or densely connected QUBOs using an iterative hybrid classical/quantum algorithm written in

C++, with MatLab and Python wrappers available.



(a) Schematic view of qbsolv internals

(b) Energy of the new solution after each trial as a function of CPU time.

Figure 3.2: qbsolv overview

As shown in Figure 3.2a, qbsolv works through trials (i.e. *main loop* in Figure). In each trial, the QUBO is split into smaller instances that are submitted to a sub-QUBO solver for global optimization. Results are combined back into one solution and a tabu search [23] is performed for local optimization. The sub-QUBO solver is ideally a `DWaveSampler`, but can also be any instance implementing the SAPI `Sampler` interface, or even a simple Python function. By default, qbsolv will use its own classical tabu solver. *main loop*

The way qbsolv breaks a QUBO into sub-QUBOs is described in [21]. The user has no control over the process, except for the number of logical qubits included in each sub-QUBO, which is fixed (`qubo_size` parameter, default to 47). When using a D-Wave as a sub-QUBOs solver, one needs to ensure that the sub-QUBOs can be embedded into the hardware (see Section 3.3). Thus, this parameter should be set by considering the maximal connectivity of a sub-QUBO. *sub-QUBOs*

The actual number of trials depends on the `num_repeat` parameter (default to 50). In a nutshell, qbsolv keeps a record of the 20 best solutions encountered so far as well as a loop counter, `RepeatPass`. At the end of each trial, `RepeatPass` is reset to zero if the new solution is the best so far or incremented if the new solution has (a) the same energy or (b) a higher energy compared to the solutions in the record. In any other case, the counter doesn't change. The main loop will stop when `RepeatPass` reaches `num_repeat`. This intricate stop criterion means that the user has no real control over the number of iterations, which highly depends on the energy landscape of the problem. Figure 3.2b depicts the energy of the new solution after each iteration for one run of qbsolv on a QUBO of 14,540 variables and 73,972 connections. Despite a `num_repeat` of 50, 117 iterations actually took place. *stop condition*

The tabu search algorithm [23] is a local neighborhood search metaheuristic: it starts from a random point on the energy landscape and visits the neighboring solutions to find a lower energy point. This has two consequences. First, the starting point can have a big impact over the final outcome. In qbsolv, this is controlled by the seed passed to the random number generator (C++ library). Second, a tabu search can get stuck in local minima. To avoid that, qbsolv keeps track of the progresses, that is the number of iterations since the last good solution found, and will restart from a new random point after 12 unfruitful passes. Those restarts explain the four peaks visible on the right side of Figure 3.2b. *tabu search*

3.6 Summary

D-Wave machines are specialized quantum annealers commercialized by D-Wave Systems that rely on the adiabatic theorem of quantum mechanics. They are believed to provide substantial speedup on a specific class of NP-Hard problems called QUBOs.

D-Wave QPUs are made of qubits connected through a sparse graph of couplers called a Chimera. Qubits are influenced by a magnetic field and their connections to other qubits. Those two mechanisms correspond to the linear components (bias weights) and the quadratic components (coupling strengths) of the QUBO.

Resolving a problem using a D-Wave has two main challenges:

1. *mapping*: expressing the problem as a QUBO;
2. *minor embedding*: transforming the logical QUBO so that it fits the Chimera of the target.

SAPI is the name of the REST API used to communicate with a remote D-Wave and submit problems. SAPI clients are part of the Ocean SDK, a suite of tools for solving QUBOs. The SDK contains powerful abstractions and leverages most of the work for us. For example, it can translate QUBOs problems into other forms (BQMs, Ising), automatically adapt the QUBO to the hardware, etc. The whole process is summarized in Figure 3.3, with automated steps shown in dash lines.

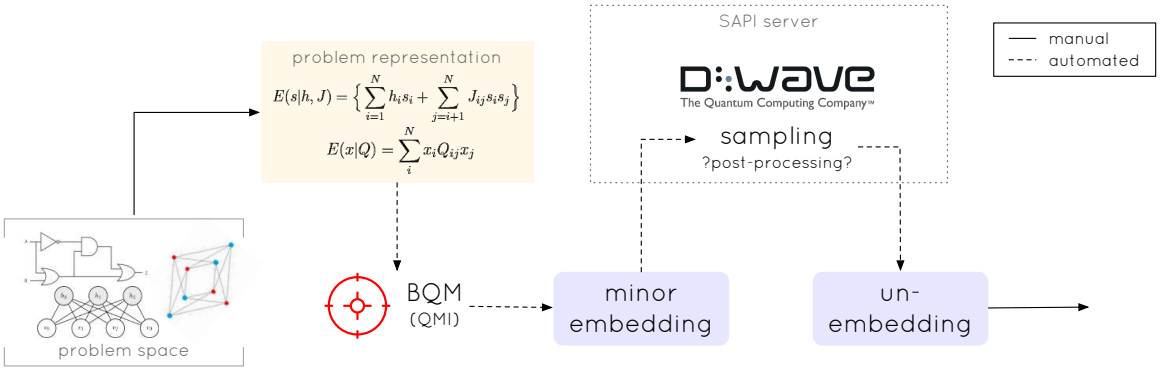


Figure 3.3: The different steps involved in solving a problem using a D-Wave

qbsolv is another tool developed by D-Wave Systems which is able to handle QUBOs too large and/or densely connected to fit the current hardware. Based on an iterative classical/quantum algorithm, it comes with a classical solver, meaning that it can be used in simulation as well. Based on heuristics, it does not, however, guarantee to find a global minima.

4 Methodology

Chapter's content

This chapter discusses a new algorithm to encode a track finding problem into a QUBO.

4.1 Previous work: Stimpfl-Abele & Garrido

In the 90s, Stimpfl-Abele and Garrido, working on the ALEPH detector at the time, experimented with different simulated annealing and Hopfield networks approaches. The work described in [24] gives a good starting point for representing a track reconstruction problem as a QUBO.

4.1.1 Overview

two tasks

By regarding "*a track with n hits as a set of $n-1$ consecutive lines* [doublets] *with a smooth shape and without bifurcation*", the problem can be broken down into two tasks:

1. Generate the set of all potential doublets S_D;
2. Find the subset of S_D which encodes the best solution.

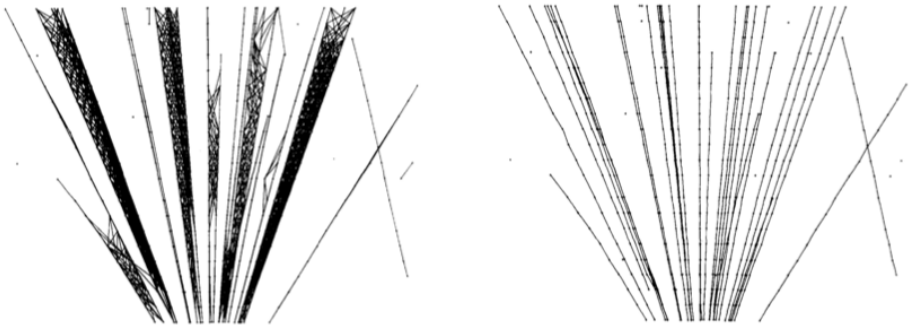
doublets generation

Generating doublets from a set of hits is a trivial task, but the size of the resulting set is quadratic. To reduce the work in step 2, doublets are generated only if the difference in ϕ is less than $0.15\,\mathrm{rad}$, the difference in θ is less than $0.10\,\mathrm{rad}$ and the doublet spans fewer than 4 detector layers. Moreover, doublets are created only in one direction, by ordering the hits in ascending order using their radius in the X-Y plane.

doublets classification

Step 2 is a binary classification problem: for each doublet, assign a 1 if kept and 0 if ignored in the final solution. An example of those two steps is shown in Figure 4.1.

(a) X-Y plane



(b) R-Z plane

Figure 4.1: Using a dataset of real $Z^0 \rightarrow$ hadrons, display of all generated doublets (left, step 1) and the kept doublets after running the binary classification task (right, step 2) [24].

4.1.2 Objective function

Hopfield network

To solve the classification task, the authors used Hopfield networks [25], an old form of recurrent artificial networks especially suited for optimization problems. Each neuron V_{ij} represents a doublet from hit i to hit j and is connected (using a non-zero weight) to neurons sharing a hit: V_{hi}, V_{jk}, V_{ik}, etc. The weights of the connections are fixed and the role of the network is to activate the right set of neurons in order to minimize an objective function. In this regard, Hopfield networks are very similar to simulated annealing.

objective function

Figure 4.2 presents the objective function of the Hopfield network, which has two main components. The first term (*connection strength*, in green) expresses the interest of activating simultaneously two doublets that are continuous (i.e. one starts at the tip of the other). This interest T_{kln} depends on the 3D angle between the doublets and the sum of their lengths: the less the angle and the length, the more interesting the connection. In other words, the idea is to favor doublets that are short and well-aligned with others, thus forming interesting tracks. The second term (red and orange) is here to enforce the rule that a hit belongs to at most one track. This constraint is expressed by penalizing the activation of two neurons that are *in conflict*. In the case of doublets, a conflict arises in two cases only: the doublets either start (red) or end (orange) at the same hit. The objective function includes a third term used to penalize the activation of too many neurons. This is problematic, because one usually has no way of foreseeing how many doublets should be included in the solution.

parameters

The objective function has two main parameters that influence the overall behavior of the network. λ is used in the connection strength computation (T_{ijk}). The literature suggests a rather large value, going from 5 up to 100, with the effect that only small angles result in a non-zero

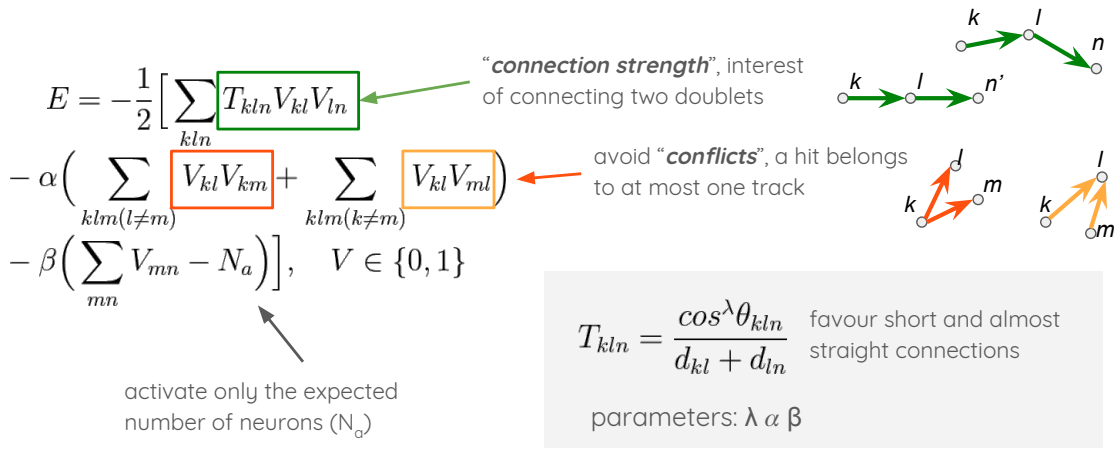


Figure 4.2: Presentation of the objective function to minimize for the Hopfield networks used in [24].

strength. α is the penalty associated to a conflict. Its value has to be set with care: if too high, an activated neuron immediately switches off all competing neurons, freezing the learning process; if too low, the constraint term vanishes and the results become unpredictable. The authors used $\lambda = 5$ and $\alpha = 5$.

Equation 4.1 shows how we expressed the objective function as a QUBO with qubits V repre- *as a QUBO*
senting potential doublets. In short, the bias weights, a, are set to zero and the coupling strengths, b, are set to either the connection strength T or the conflict penalty α.

$$O(a;b;V) = \sum b_* V_{ij} V_{kl}, \quad b_* = \begin{cases} -T_{ijk}, & \text{if } j == k \\ \alpha & \text{if } i == k \vee j == l, \\ 0 & \text{otherwise.} \end{cases} \quad V \in \{0,1\} \tag{4.1}$$

4.1.3 Interest of the approach

Using several datasets from the ALEPH detector, the time and physics performances proved *performance*
to be competitive with the conventional methods used at the time. The memory requirements are low and the efficiency is above 99% for tracks of 6 hits or more.

However, this approach also has its limitations: *limitations*

1. The only way to limit the number of potential doublets to a fair amount is to make assumptions on the location of the primary vertex: the ϕ and θ angles used by the authors in step 1 only make sense if the track actually comes from the origin ($(0,0,0)$).
2. Using only doublets pair, it is difficult to ensure a good continuity along a track. Coupled with a high α value, this often results in zigzag patterns instead of two rather close parallel tracks; this is even more true as the density of the dataset increases.
3. The cuts made during early selection as well as the definition of an interesting connection as short and straight only works for high p_t particles with a very low curvature in the transverse plane.
4. The results have more false positive (i.e. wrong tracks) than conventional methods, resulting in a lower purity.

Hence, the Stimpfl-Abele approach is very interesting for our purpose, but suffers some strong limitations, amongst which a low purity (Item 4) and a bad scalability (Item 2): in the paper, the authors report a good performance for datasets up to 200 tracks and < 4000 hits.

4.2 Algorithm goals

goals

Following [24], we view the pattern recognition task as a doublet classification problem: given a set of potential doublets, find the subset of doublets that compose the real track candidates. This corresponds to Step 2 in Subsection 4.1.1). As an additional goal, the classification must be encoded as a QUBO and run on a D-Wave. The algorithm is developed with the TrackML dataset in mind and targets the TML detector (see Section 2.4).

high p_t assumptions

The study relies heavily on the assumptions that the particle tracks of most interest are the high P_t tracks and that, neglecting noise and multiple scattering, they exhibit the following properties:

- the hits draw an arc of a helix in the transverse plane, with a very small radius of curvature;
- the hits are aligned in the R-Z plane, forming a straight line;
- there are few to no holes (missing hits): most hits lie on consecutive layers.

Note that those properties can be computed for any set of 3 or more hits and don't rely on any vertex assumption.

other assumptions

Additionally, we find that track candidates with fewer than 5 hits are predominantly fake tracks or bear minimal interest for study. Finally, although it is physically possible for tracks to share hits, we stick to the constraint from the TrackML challenge that any one hit can belong to at most one track.

4.3 Triplets and quadruplets

To avoid some of the caveats discussed in Section 4.1.3, the algorithm uses triplets instead of doublets as the main unit of work.

4.3.1 Definitions

definitions

A triplet, denoted T^{abc}, is a set of three hits (a, b, c) or a pair of consecutive doublets (a, b and b, c), ordered by increasing transverse radius (R). Two triplets T^{abc} (of hits a, b, c) and T^{def} (of hits d, e, f), can be combined to form a quadruplet if $b = d \wedge c = e$ or a quintet if $c = d$. If they share any other hit, they are marked as having a conflict. A track candidate of n hits is constructed from a set of $n - 2$ triplets that can be combined into $n - 3$ quadruplets. In the remaining of the report, we'll use the term xplet to refer to subsets of hits such as triplets and quadruplets. This is exemplified in Figure 4.3.

advantages

Using triplets and quadruplets yields many advantages. First, it allows to weaken assumptions about particles originating from the primary vertex. Second, it makes zigzag patterns less probable, since zigzagging quadruplets won't be generated at all. Finally, they exhibit more physical properties that can be exploited in the formulation of the QUBO (see next section).

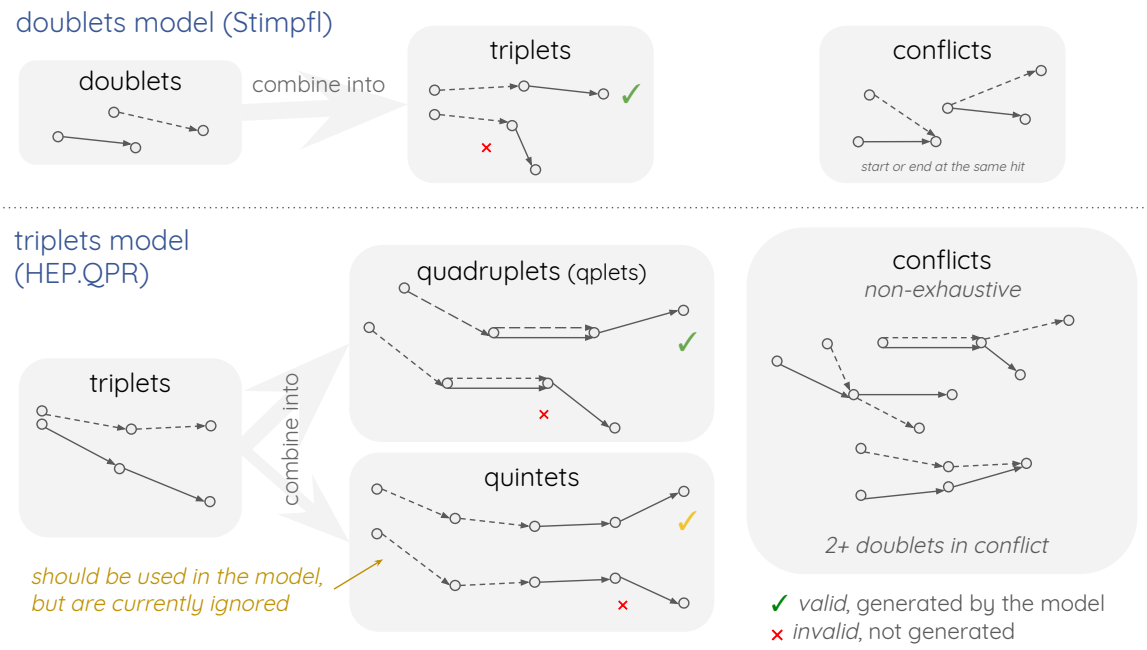


Figure 4.3: Examples of triplets, quadruplets and quintets and comparison between the doublets VS the triplets model.

4.3.2 Xplet quality

Xplets can be more or less interesting. The quality of an xplet is used both for preselection (*quality cuts*) and for the generation of the QUBO.

The quality of a triplet T_i^{abc} is defined using three properties: *triplet quality*

1. the number of holes, H_i: ideally, the hits are in consecutive layers and thus $H_i = 0$;
2. the radius of curvature [26] in the transverse plane, $curv_i$;
3. the difference of angles formed by the two doublets in the RZ-plane: $drz_i = |\delta(\theta_{ab}, \theta_{bc})|$, with $\theta_{ab} = tan^{-1}(\frac{\delta z}{\delta r})$.

Similarly, the quality of a quadruplet (T_i, T_j) can be calculated from the compatibility between the triplets. The strength S is used to quantify how well they satisfy the high P_t track properties. This is similar to the *connection strength*, T, in Stimpfl-Abele's objective function (see Section 4.1.2): *quadruplet quality*

$$S(T_i, T_j) = \alpha \frac{\beta\big(1 - |\delta(curv_i, curv_j)|\big)^{\gamma} + (1-\beta)\big(1 - max(drz_i, drz_j)\big)^{\delta}}{(1 + H_i + H_j)^{\epsilon}} \tag{4.2}$$

where β encodes the relative importance of the curvature with respect to the Z-R angle and the other parameters are unbounded constants that require problem-specific tuning. In its simplest form, $\beta = 0.5$ (equal weights), $\epsilon = 2$ (norm), and all other constants are set to 1:

$$S(T_i, T_j) = \frac{1 - \frac{1}{2}(|\delta(curv_i, curv_j)| + max(drz_i, drz_j))}{(1 + H_i + H_j)^2} \tag{4.3}$$

4.4 Quality cuts

To limit the size of the QUBO, cuts are applied on the xplets using the quality criteria described above. Additionally, the input doublets are also filtered based on the number of holes. By default, no more than one hole (i.e. one potential missing hit) is allowed.

Table 4.1 presents two sets of quality cuts for triplets and quadruplets. The first one does not favor any particular momentum range, resulting in large QUBOs. The second one focuses on high P_t tracks ($p_t >= 1\,\mathrm{GeV}$) relevant for physics analysis at the HL-LHC. *two sets of quality cuts*

	Low p_t cuts	**High p_t cuts**
Triplets		
Number of holes, H_i	≤ 1	≤ 1
curvature, $\lvert curv_i \rvert$	$\leq 5 \times 10^{-3}$	$\leq 8 \times 10^{-4}$
delta R-Z angle, drz_i	≤ 0.2	≤ 0.1
Quadruplets		
delta curvature, $\lvert \delta(curv_i, curv_j) \rvert$	$\leq 4 \times 10^{-4}$	$\leq 1 \times 10^{-4}$

Table 4.1: The two sets of quality cuts used during preprocessing to limit the number of xplets and thus the size of the QUBO.

In addition to the quality cuts, two other mechanisms are used to limit the number of xplets: *additional filtering*

1. To be selected, a quadruplet must be part of at least one chain of `<max_qplet_path>` or more quadruplets; the default value, `max_qplet_path=2`, means that each quadruplet has the potential to create at least one track candidate of 5 hits or more;
2. In a final pass, doublets and triplets that are not part of at least one quadruplet are discarded.

Figure 4.4 summarizes this process. The quality cuts are applied during xplet creation (1st pass). The other mechanisms require a 2nd pass. Only the remaining xplets are used in the QUBO. *timing*

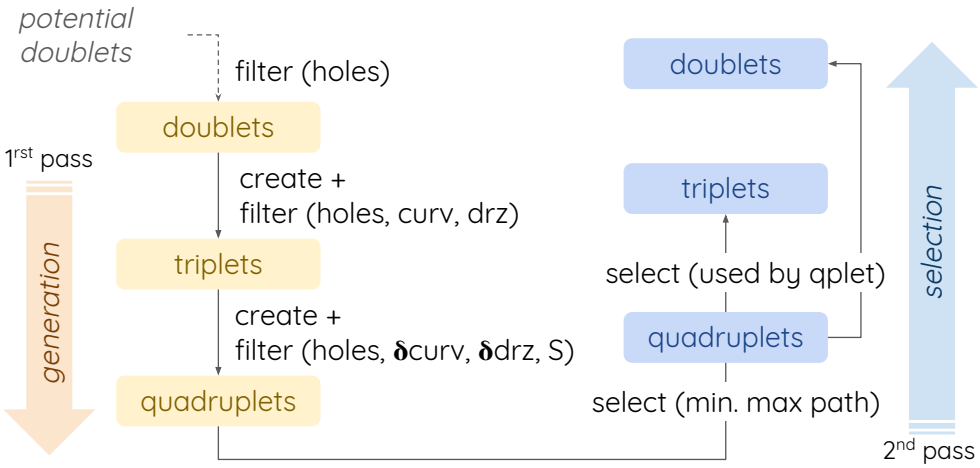


Figure 4.4: Steps involved in the creation and selection of the xplets.

4.5 The Quadratic Unconstrained Binary Optimization

the QUBO

The goal of the objective function is to select the best set of triplets, in the form of pairs, that constitute valid track candidates with no conflicts. The mathematical formula is given by:

$$O(a, b, T) = \sum_{i=1}^{N} a_i T_i + \sum_{i}^{N} \sum_{j<i}^{N} b_{ij} T_i T_j \quad T \in \{0, 1\} \tag{4.4}$$

where T are all potential *triplets*, a_i are the bias weights, and b_{ij} the coupling strength computed from the relation between the triplets T_i and T_j:

$$b_{ij} = \begin{cases} -S(Ti, Tj), & \text{if } (T_i, T_j) \text{ form a quadruplet,} \\ \zeta & \text{if } (T_i, T_j) \text{ are in conflict,} \\ 0 & \text{otherwise.} \end{cases} \tag{4.5}$$

the a_i parameter

As quality cuts have already been applied (see Section 4.4) and to simplify the model, we decided to consider the triplets included in the QUBO as equally likely. Consequently, the bias weight a_i is set to a constant a and the selection is driven by the coupling strength. The actual value a to use, however, will have to be tuned. A priori, setting $a = 0$ seems logical, as it doesn't state anything about the triplet prior. However, the use of a small positive or negative value ($a \in \{-0.1, 0.1\}$) shouldn't be overlooked as this can change the energy landscape and thus impact both the physics and timing performances of the internal QA process[1].

the ζ parameter

Similar to the α parameter in Equation 4.1, the ζ parameter, or *conflict strength*, must be set with care. Generally speaking, it should be high enough to counterbalance the connection strength S, but not too high in order to avoid introducing discontinuities in the energy landscape.

need for testing

In the end, the different parameters are all interdependent and the same set of values can behave differently given the QUBO solver in use. Consequently, only rigorous experiments can help determine the best sets of values to use.

4.6 Post-processing

tasks

The final steps of the algorithm include determining the kept triplets from the QUBO solution, converting the triplets back into doublets and computing the various metrics.

QUBO solution

As described in Section 3.5.1, samplers available in the Ocean suite return multiple answers for one QUBO. We decided to systematically pick the answer with the lowest energy, as opposed to the most recurrent one.

conflicts

Converting triplets back into a set of doublets is trivial. However, due to the inherent imperfections of the available QUBO solvers and the choice of constants, conflicts may remain in the final solution. In this case of figure, we developed two strategies:

1. ignore conflicts by removing all conflicting doublets from the solution;
2. try to resolve conflicts by first recreating track candidates from the set of non-conflicting doublets, then adding conflicting doublets one-at-a-time in order to maximize the length of the track candidates, stopping when no more doublet can be added.

[1] This is even more important when using a tool like qbsolv (see Section 3.5.2), which relies on multiple iterations and convoluted convergence criteria.

Early experiments showed that the second strategy is not subtle enough to guarantee good results: in general, half of the conflicting doublets added are fakes. Finding a better strategy is thus an area to work on.

track length

When using a `max_path > 1` (see Section 4.4), the algorithm shouldn't be able to find track candidates with less than a given number of hits (x, 5 by default). If short tracks are present in the results, they have a great probability of being fakes or hazardous. For this reason, they are systematically ignored in the final solution. Said otherwise, only doublets part of track candidates of x hits or more are included in the final solution.

4.7 Algorithm overview & summary

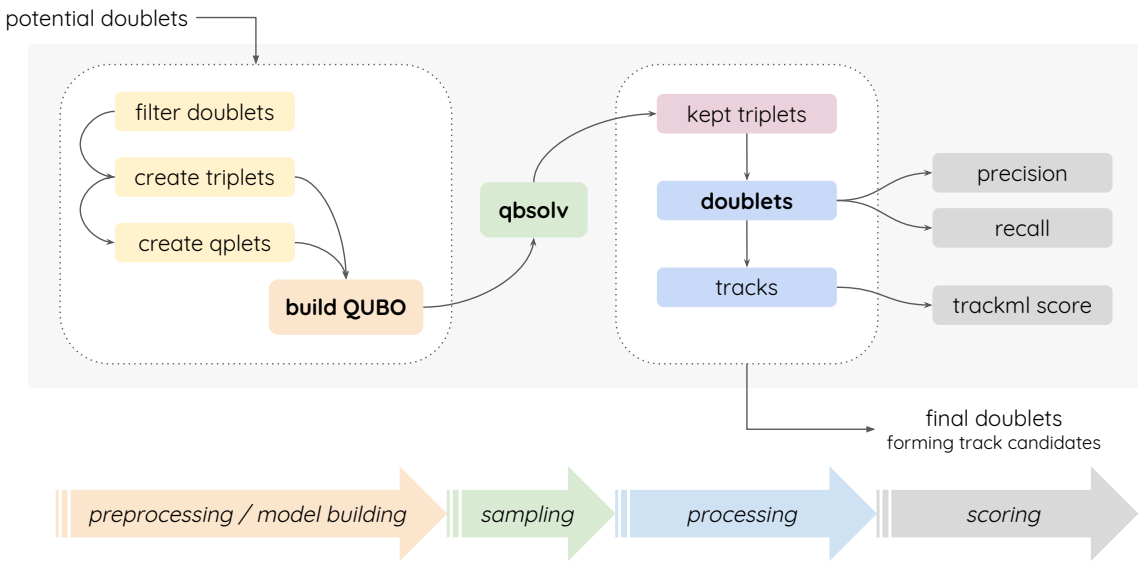


Figure 4.5: Overview of the algorithm flow.

Figure 4.5 gives an overview of the complete algorithm. The initial doublets are combined into triplets and quadruplets, after satisfying the requirements from Section 4.4. The QUBO is generated (Section 4.5) and sampled using the qbsolv library (Subsection 3.5.2). The post-processing phase includes converting the triplets into doublets, removing duplicates and dealing with potential remaining conflicts. The track candidates are reconstructed from the doublets, and doublets from candidates with less than 5 hits are discarded (Section 4.6). Finally, performance metrics are computed and the set of final doublets is returned.

5 Experimental setup

Chapter's content

This chapter presents the datasets and metrics used for the experiments, as well as the general benchmark procedure used throughout this study.

5.1 Datasets

simplifications

Events from the TrackML dataset (see Section 2.4) have been simplified by removing:

1. hits from the end-caps;
2. double hits.

barrel only

The end-caps are always challenging to work with. First, it makes it more difficult to define *consecutive* layers. Second, there is no simple way to orient doublets, as the particles can draw arcs of helices that curve back toward the center of the detector in the transverse plane. For those reasons, only barrel hits are currently supported[1] and extending the algorithm to end-caps is left for a future iteration.

no double hits

Double hits are hits left by the same particle on the same layer, but on different modules. As such, they don't convey much information. However, they still represent around 15% of the hits in a TrackML event. We think they could be detected in a pre-processing step or dealt with in post-processing, so they are ignored in the main algorithm.

smaller datasets

In order to test the algorithm on different occupancies, smaller datasets are created from the full events by randomly sampling both the particles and the noise. A dataset-P% is defined by the selection of P% particles and P% noise hits. The resulting dataset should thus conserve similar characteristics, with the notable exception of the hit-to-noise ratio that can be a bit lower when $P < 100\%$.

[1] This cut was suggested by our ATLAS supervisor; focusing on the barrel is a common simplification in the field.

focused datasets

Unless stated otherwise, the focus is drawn on high p_t particles with at least 5 hits (focused datasets). That is, hits from low p_t (<1 GeV) and/or short particles are kept in the dataset, thus part of the pattern recognition, but they are not taken into account when computing the performance metrics (see next section).

TrackML weights

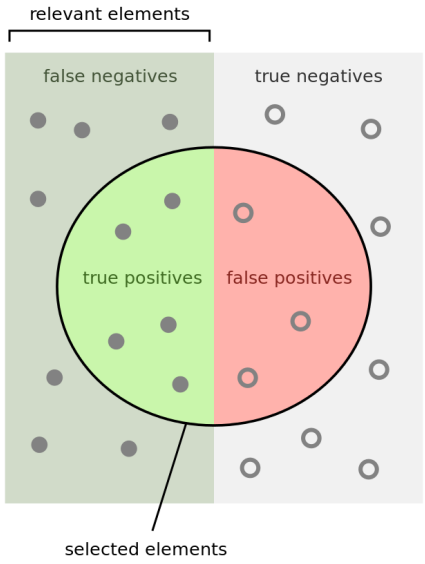
As described in Subsection 2.4.4, the computation of the TrackML score uses weights associated with hits. When creating subdatasets, those weights are updated in the following way:

- for focused datasets, the weights of hits left by low p_t or short particles are set to 0;
- the new weights are normalized to guarantee a sum of 1, as for the original event ground-truth.

5.2 Metrics

trackml, precision, recall

The metrics used are the TrackML score (Subsection 2.4.4), the purity and the efficiency. The terms purity and efficiency are commonly used in collider physics and especially in track reconstruction tasks, but lack a precise definition. In this work, we treat them as synonyms to *precision* (p) and *recall* (r), as defined in Figure 5.1.



Let's call $D^\star$ the set of true doublets (TP+FN) and D the final set of doublets selected by the algorithm (TP+FP).

The purity, or precision, $\boldsymbol{p}$ answers the question: *how many selected items are relevant ?*

$$p = \frac{|D \cap D^\star|}{|D|} = \frac{TP}{TP + FP} \tag{5.1}$$

The efficiency, or recall, $\boldsymbol{r}$ and answers the question: *how many relevant items are selected ?*

$$r = \frac{|D \cap D^\star|}{|D^\star|} = \frac{TP}{TP + FN} \tag{5.2}$$

Figure 5.1: Definition of purity, efficiency, precision and recall

true doublets

Determining the set of true doublets $D^\star$ from a set of particle hits (as available from the truth files of the TrackML dataset) is not trivial, because to reconstruct the doublets, one has to *order* the hits. To simplify, we decided to systematically order hits by increasing radius R in the transverse plane. This usually works well for hits in the barrel, but can lead to wrong doublets when end-caps hits are also considered. An example is shown in Figure 5.2.

p & r in focused datasets

Generally speaking, generated doublets fall into two categories: `real` or `fake`. However, when focusing on specific particles, such as high p_t, the `real` category is split into two subsets: (A) real doublets belonging to an interesting particle track; (B) real doublets belonging to a particle track not under focus (low p_t, too short, etc.). In such cases, generated doublets from category (B) are set aside *before* computing the precision/recall and the set of true doublets ($D^\star$) includes only true doublets from interesting particles.

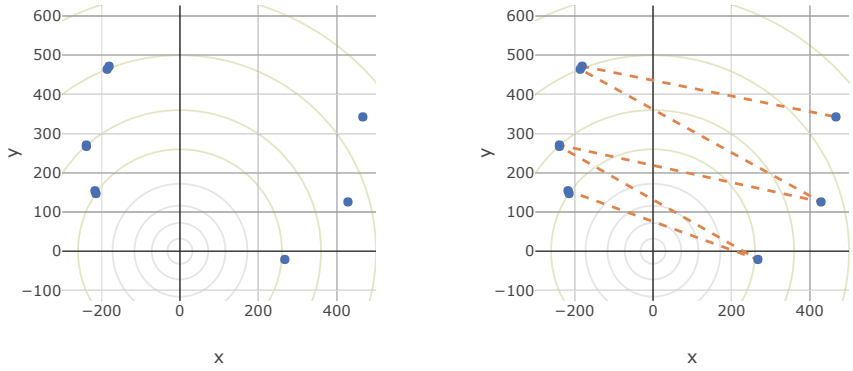


Figure 5.2: Example of problematic particle: ordering hits by increasing radius in the transverse plane leads to wrong doublets (right). The circular marks show the location of the barrel layers.

5.3 Initial doublets generation

original code

The generation of the initial doublets uses a Python code derived from [27], which is itself an adaptation of the online seeding GPU code used in the ATLAS Trigger system [28].

modifications

Aside from making the code compliant with the TML detector, modifications were made to limit the task to doublet generation and to improve the overall efficiency. As a result, the program can generate doublets from full events in a decent amount of time and with a recall above 95% for high p_t tracks. The trade-off, however, is a very low precision that can drop below 1% for big datasets. The code was also rewritten to provide a clean API and a command-line interface. Among the options, there is the possibility to ensure a recall of 100% by *cheating*, i.e. by peeking at the ground-truth, but it was never used in benchmarks.

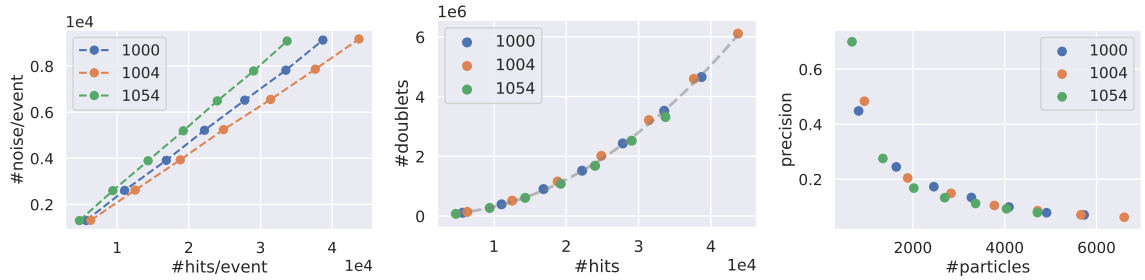
layers

Note that the seeding code looks at the length of a doublet and not the number of layers it covers. As a result, many doublets generated during the seeding will be discarded early in the model building step of the algorithm (see Section 4.4).

5.4 Test sets

21 test sets

From the principles described in Section 5.1, a test set of 21 focused datasets have been created and used across all benchmarks.



(a) Noise-to-hits ratio.

(b) Hits and input doublets.

(c) Input precision.

Figure 5.3: Benchmark datasets statistics.

sources

The datasets were generated from 3 TrackML events, 1054 (small), 1000 (medium) and 1004 (the largest available), using percentages of whole decimal number from 10% to 70%. Note that the biggest dataset, evt1004-70%, has more particles than some of the smallest TrackML events.

occupancy

The occupancy varies between 673 and 6604 particles/event. The ratio between the total number of particles and the focused particles is stable accross datasets ($\sim 8\% \pm 0.6$). The noise-to-hits ratio is around 23% ($\sim 23.8\% \pm 3$), datasets from event 1054 being the noisiest (see Figure 5.3a).

input doublets

The input doublets were generated once and saved in a CSV file. The input precision stays below 0.7% (see Figure 5.3c), while the recall is always above 98%. As shown in Figure 5.3b, there is a superlinear relationship between the number of hits and the number of input doublets.

More details are available in Appendix A.

5.5 Experiments overview

two steps

The experiments were split into two parts: (A) Model building (i.e. QUBO generation, "build"), (B) QUBO solving & post-processing ("run"). Step (A) was run only once for each dataset and the generated QUBO serialized to disk. This ensures all run experiments are comparable and also spares time, as model building takes a substantial portion of the total runtime.

parameters

Unless stated otherwise, parameters were left to their default values and the QUBOs were generated using $a = -0.1$ and $\zeta = 1$. The quality cuts applied during model building are those for high p_t tracks (see Section 4.4). When using qbsolv with a D-Wave, we also limited to 10 the number of reads (i.e. how many times the anneal is performed) and the maximum iterations (see `MaxRepeat` in Section 3.5.2).

environment - small runs

Small runs and early experiments were run on a 2016 MacBook Pro Laptop with 16 GB of RAM using tools such as PyCharm [29] and Anaconda/Jupyter [30] and the D-Wave 2000Q at Oak Ridge National Laboratory available from the Leap Platform [19].

environment - large runs

For larger experiments and benchmarks, model building and runs in simulation (i.e. no D-Wave) ran on a Haswell node from the Cori supercomputer at NERSC [31]. D-Wave experiments used the Ising D-Wave 2X machine at Los Alamos National Laboratory. Because of the security processes in place, they had to be launched from a Linux machine with 32 CPU cores and 50 GB of RAM instead. However, since qbsolv is not parallelized in any way, this shouldn't impact the validity of the comparisons.

6 Results

Chapter's content

This chapter presents the main results of the study using the datasets and procedures described in Chapter 5. For the latest results, see also Section 7.2.3.

6.1 Overview

To get started, the following subsections describe two complete runs (MacBook) of the algorithm using small focused and non-focused datasets.

6.1.1 High p_t run

dataset & input

Figure 6.1 presents the results of the algorithm on a focused dataset-20% generated from event 1000 with 1637 particles and 11,030 hits. Only 145 particles are of interest. The seeding (see Section 5.3) created 392,529 input doublets, with a recall of $r = 99.15\%$ (9 missing high p_t doublets) and a low precision of $p = 0.27\%$.

	Input	Created	Used	Missing
doublets	392,529	196,260	2546	9
triplets	-	91,484	2964	3
quadruplets	-	3631	3051	5

Table 6.1: Number of xplets created (1st pass), used (2nd pass) in the QUBO (see Section 4.4) and real xplets not created due to the quality cuts used.

QUBO generation

During model building, the high-p_t cuts discarded more than 99% of the initial doublets (see Table 6.1). The final QUBO uses 2964 variables and 11,381 couplers, 8330 of which being the expression of a conflict. The whole model building took less than 1 min.

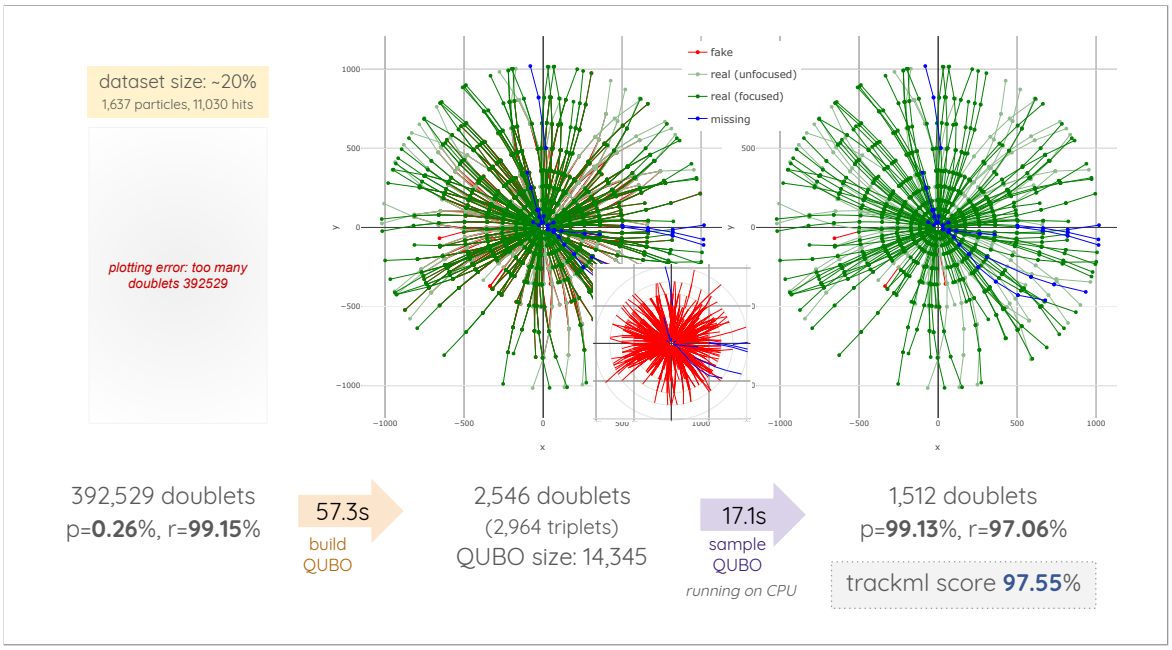


Figure 6.1: Overview of the results on a focused dataset of 1637 particles.

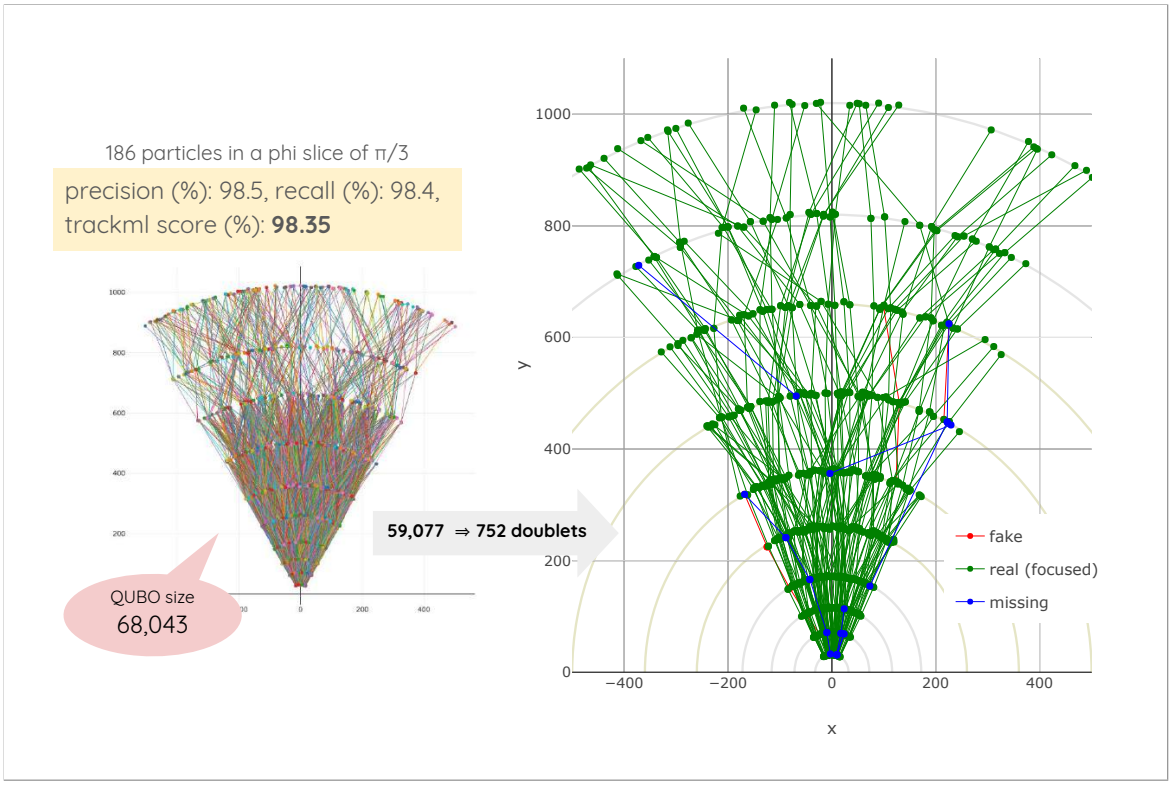


Figure 6.2: Overview of the results on a dataset of 186 particles. All particle tracks with at least 5 hits (125) are considered of interest (no p_t focus).

QUBO solving

Qbsolv in simulation mode with default parameters selected 1305 triplets (∼60%), with no remaining conflicts. The energy of the best sample was −1073.5626, which is slightly below the energy of the ideal solution (−1068.1267). This highlights some of the limitations of the QUBO formulation, as the system was able to find a better solution by selecting fakes.

The final results are very good, with all metrics above 97% for the particles of interest (9 fakes and 31 missing doublets, 211 track candidates). It is also worth mentioning that some of the low p_t particle tracks were recovered as well (light green doublets in Figure 6.1). *final results*

6.1.2 Low p_t run

Figure 6.2 presents the results of the algorithm on a small dataset of 186 particles located in a compact ϕ slice. Here, all particles with at least 5 hits (125) are of interest. The low p_t quality cuts were used during model building and the QUBO constants were set to $a = 0.1$ and $\zeta = 0.8$. *dataset & QUBO*

As we can see, the algorithm works well for low p_t tracks as well, with a final TrackML score above 98%. However, in this setting, the size of the QUBO is very large: 3590 variables and 64,453 couplers, 61,009 of which representing conflicts. This is more than four times more parameters than in the previous example, despite a number of particles two orders of magnitude smaller. *results & comparison*

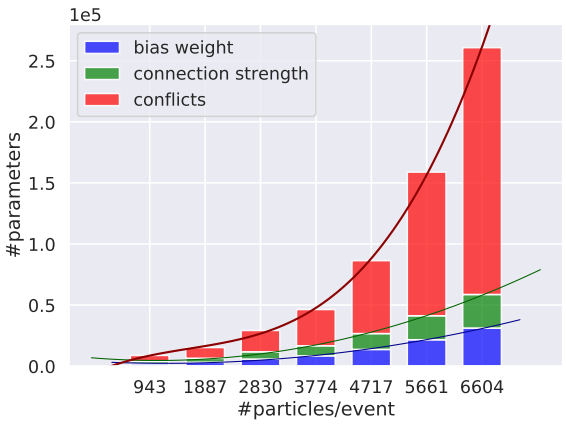
6.2 Model building performances

The main step of model building is the creation (1st pass) and selection (2nd pass) of the xplets (see Figure 4.4). As shown in Table 6.2, the 2nd pass reduces drastically the number xplets actually used in the QUBO. The filtering of the input doublets based on the number of holes is also worth the effort: allowing only one missing layer discards in average 50% of the input. *quality cuts*

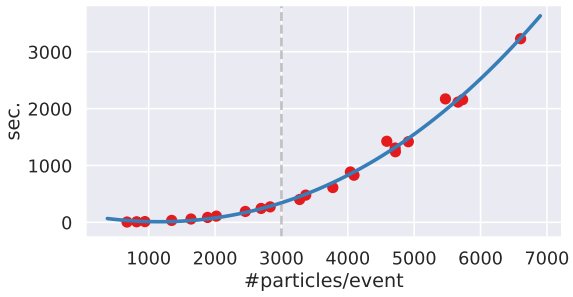
doublets	triplets	quadruplets
1.16 ± 0.63	2.53 ± 3.47	55.51 ± 23.79

Table 6.2: Ratio used/created during model building (in %, averaged).

The size of the generated QUBO is driven by the number of conflicts, which amounts to more than 60% of the total number of coefficients and increases at least quadratically with the occupancy (see Figure 6.3a). *QUBO size*



(a) Size and composition of QUBOs (event 1004).



(b) Build time (Wall).

Figure 6.3: Build time and composition of the QUBOs as a function of occupancy.

The higher the occupancy, the closer the hits and the more the potential xplets. As a result, the build time raises superlinearly with the number of particles per event (remember from Section 5.3 that the number of input doublets also increases superlinearly). However, from 3000 particles onward, the curve looks more and more like a straight line (see Figure 6.3b). The wall time varies from *build time*

5 s up to 54 min for the largest dataset of the benchmark. Using profiling tools such as vmprof [32], we identified one clear bottleneck to be the combination of doublets into triplets.

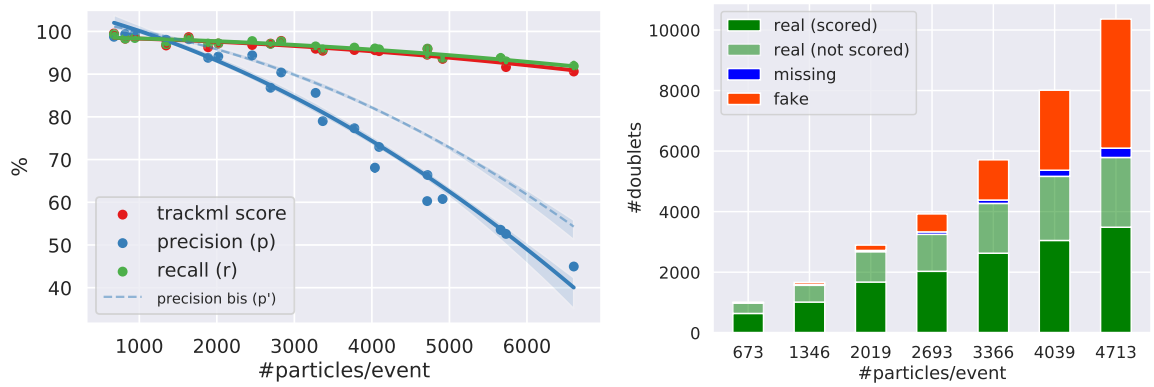
building time improvements

It is, however, important to stress that the current implementation of the model building targets ease of development and not performance; it has not been optimized in any way and uses only one CPU. Moreover, the code includes a lot of overhead in order to keep track of the processing and to gather metrics. As an example, each combination of doublet is checked against the truth and the result potentially logged in a file. We have strong hope that by removing the overheads, improving the seeding, changing the data structures and taking advantage of tools such as NumPy, the building time may be greatly improved.

6.3 Physics performances

This section presents the overall performance of the algorithm in terms of physics. Simulation results are discussed first, followed by a comparison with the D-Wave runs.

6.3.1 Performances in simulation



(a) Scores as a function of occupancy.

(b) Composition of the final doublets.

Figure 6.4: Physics performance in simulation (qbsolv+classical)

metrics

As shown in Figure 6.4a, the TrackML score and the recall stay steady and remain above 90% across all datasets. The precision, however, drops rather quickly as the occupancy rises and falls below 50% at around 6000 particles/event. The same conclusion can be drawn from Figure 6.4b, which shows the kind of doublets composing the output for a subset of occupancies.

non-focused doublets

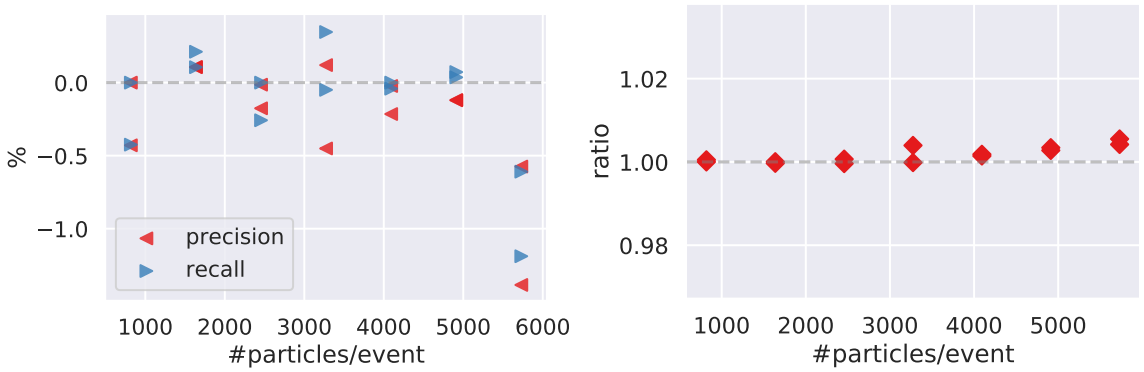
As the occupancy increases, we find more and more doublets belonging to non-targeted particles (light-green in Figure 6.4b). For large occupancies, they account for more than 30% of the output. If we take them into account to compute a new precision p' (dash line in Figure 6.4a), we notice an improvement up to 12%. Note, however, that they represent only $10\% \pm 0.71$ of the total number of true non-focused doublets (counting short tracks).

energies

Given a QUBO and the ground truth, it is trivial to compute the energy of the ideal solution, en_0. For very small datasets, the difference between the ideal and qbsolv energies is very small. As the occupancy rises, the gap increases towards lower energies (see Appendix B.2). This again points to the fact that the fake rate is a shortcoming of the model itself: the better the tool at minimizing the QUBO, the more the fakes.

6.3.2 Performances using a D-Wave

Due to time constraints, D-Wave runs were only performed twice on datasets derived from event 1000. To improve the comparisons, qbsolv was initialized with the same seeds as in simulation. *data*



(a) Difference in precision and recall (D-Wave - sim.). A point above 0 means a better D-Wave result.

(b) Difference in energies (ratio sim./D-Wave). A value above 1 means a better D-Wave result.

Figure 6.5: Comparison between simulation and D-Wave runs using qbsolv.

In general, we couldn't see any substantial difference in the behavior of qbsolv. The physics performance is very similar, the scores varying by less than 1% overall. Still, it is interesting to note that the energies returned using D-Wave are always slightly better (i.e. lower, see Figure 6.5b), but this does rarely translate into better performances (see Figure 6.5a). *comparison*

6.4 Timing performance

This section discusses the total runtime of the algorithm and some of the difficulties of performing timing benchmarks on a D-Wave.

6.4.1 Total runtime (simulation)

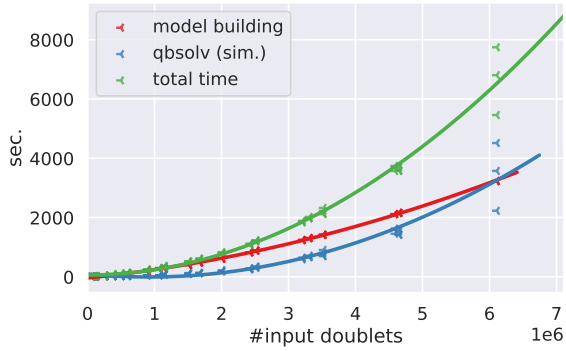


Figure 6.6: Wall time as a function of the input size in simulation mode.

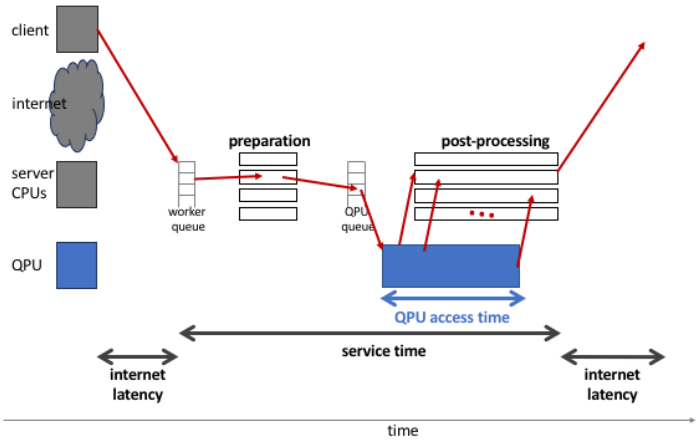
Figure 6.6 shows the time performance of the algorithm using qbsolv in simulation. The total time comprises post-processing as well, but it is not plotted as it never exceeds 0.5 s. Overall, the runtime is superlinear as a function of occupancy. Model building dominates the total runtime for *overview*

low occupancies, up until around 6000 particles/event where qbsolv starts to struggle. The largest run took up to 2 h to complete.

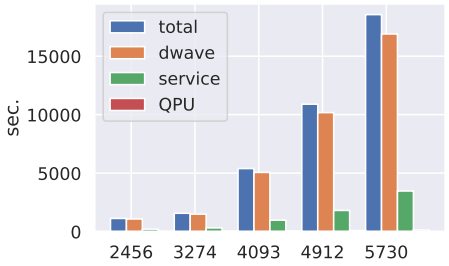
Figure 6.6 also highlights the impact of the random seed on qbsolv (most probably linked to the stop criterion described in Section 3.5.2). On the biggest event, the difference between runs is above 20 min – 20% of the total time. *qbsolv seed*

6.4.2 Timing on qbsolv/D-Wave

D-Waves are remote computers shared across users. As a result, the runtime is longer and can vary unexpectedly depending on the internet latency, the number of users currently logged in, the security in place, etc. making comparisons difficult. *time overview*



(a) Overview of execution of a single QMI, starting from a client system, and distinguishing classical (client, CPU) and quantum (QPU) execution [33].



(b) Overview of the runtime of qbsolv using D-Wave for a subset of occupancies.

Figure 6.7: D-Wave runtime overview.

Aside from the communication and sharing overheads, there is also the setup of the D-Wave machine, the pre- and post-processing of the raw results and the minor embedding/unembedding that has to be performed for each sub-QUBO. The latter is done on the client machine and during the experiments, we noticed that it often failed (exceptions raised from minorminer), resulting in more overhead.

Figure 6.7 gives an overview of the qbsolv timings using D-Wave. The internet latency is clearly dominating the runtime (difference between the orange and the green bars). The service time is also important and seems to take longer than the purely classical qbsolv steps (difference between the blue and orange bars). The QUBO sampling took up to 5h40min to complete, against a maximum of 30 min in simulation. *results*

Even by subtracting all the D-Wave overhead (i.e. keeping only the difference between the blue and orange bars and adding the QPU time), the runtime is longer: for 5730 particles/event, it takes an average of 1750 s, against 1500 s in simulation (or 12 s, see Subsection 6.5.2). This tends to show that even under ideal conditions, there is no possible time gain in using a D-Wave[1]. *no advantage*

See Appendix C for a more detailed discussion of D-Wave timing.

[1] This is at least true as long as qbsolv is needed to split the QUBO.

6.5 Other experiments

Many other experiments have been performed during this study. This section briefly presents a small selection that may be of interest to the reader.

6.5.1 qbsolv sub-QUBO size

expectations

qbsolv's `solver_limit` parameter controls the number of variables used in each sub-QUBO. Since the D-Wave anneal time is supposed to be constant as long as a problem fits the hardware, we expect larger sub-QUBOs to improve the runtime with no impact on the physics performance.

experiment

The experiment was carried on two datasets of small to middle occupancy (10% and 40% from event 1000) using 5 different solver limit: 47, 60, 100, 150 and 200.

results

On the small dataset, the runtime varies unexpectedly. On the larger dataset, increasing the solver limit divides the runtime by up to a factor 2. The best speedup was obtained with a solver limit of 150. As expected, there is no significant impact on the quality of the physics performance, which varies by less than 1%.

oddity

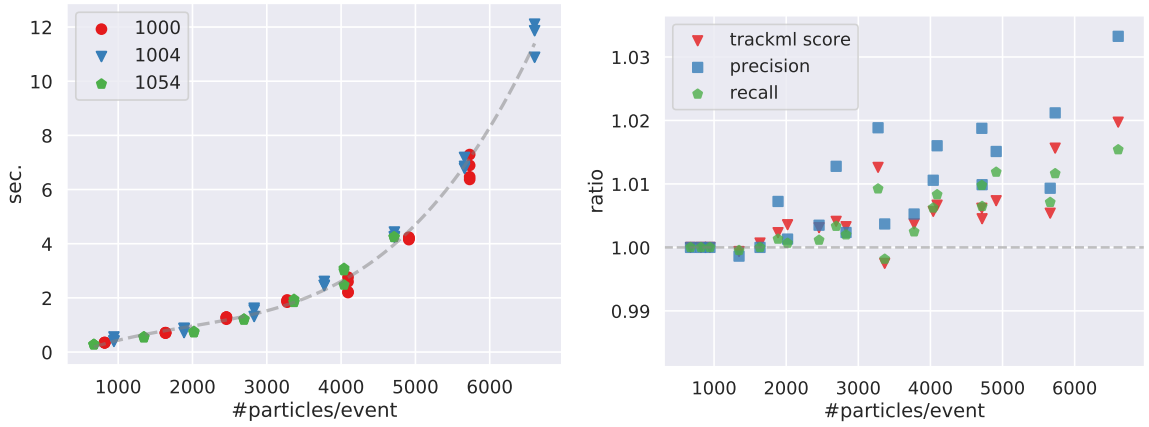
A speedup can easily be explained by the reduction of overhead (D-Wave setup, service time, etc.). However, we didn't expect a limit of 60 to be 1.2x faster than 200. Actually, this difference can be explained when looking at the behavior of qbsolv. For a solver limit of 200, it took 126 more iterations to converge (1.14x more). An added factor could be the greater difficulty of embedding larger sub-QUBOs, but we didn't confirm this hypothesis.

See Appendix B.1 for additional figures.

6.5.2 Another QUBO solver (neal)

about neal

In order to have a point of comparison, all the experiments have been reproduced using another classical QUBO solver. Out of the many alternatives available, neal [34] was chosen as it is free and offers an interface compatible with the D-Wave Ocean SDK.



(a) Wall Time performance (sampling only)

(b) Physics performance compared to qbsolv (sim.)

Figure 6.8: Runtime and physics performance using neal as the QUBO solver.

The sampling time using neal is superlinear as a function of occupancy and shows a curve similar to qbsolv. However, the time scale is reduced by more than two orders of magnitude. As shown Figure 6.8a, neal takes no more than 12 s on the largest dataset, compared to up to 37 min with qbsolv. Figure 6.8b also shows slightly better physics performance, especially as the occupancy rises. It is important to stress, however, that we are talking about less than 1% improvements on the final metrics. Thus, the interest really lies in the time gain. *100x faster*

Another benefit of neal is that it seems relatively insensitive to the random seed used. It provided stable results and runtime across all experiments, making it more predictable than qbsolv. *stability*

6.5.3 QUBO parameters

To assess the impact of the QUBO constants, experiments were conducted on two events (1000 and 1004) with neal as the sub-QUBO solver. The tested values were $a \in \{-0.1, 0, 0.1\}$ and $\zeta \in \{0.8, 1, 2\}$. Each combination has been tested three times on each dataset using different seeds. *experiment*

The physics performance is not really impacted, with standard variations smaller than 2.3% within each dataset and for all metrics. The runtime stays below 13.3 s for all experiments. The variations in runtime are more visible as the occupancy rises, but seem to be caused by the heuristic nature of the algorithm and not by the parameters *per se*. Given those results, we decided it wasn't worth investigating further. *results*

However small, one should note that the default parameters used in benchmarks, $a = -0.1$ and $\zeta = 1$ appear suboptimal. A value of $a = 0.1$, for example, seems to consistently improve the metrics by a small factor ($\sim 1 - 3\%$). For ζ, smaller values yield better results, but we found that a value below 1 leaves conflicts in the solution. See Appendix B.5 for more details. *default parameters*

6.5.4 Run on full events

The main experiments were also run on the two smaller events of the TrackML training set, 1062 and 1039. After removing the end-caps and double hits, the occupancy is respectively 4587 and 5471 particles. This is roughly equivalent to 60-70% of event 1004. *datasets*

The achieved performance is slightly worse than expected. The TrackML score and recall are just above 90%, while the precision drops to 49% on event 1062 and as low as 45% for event 1039. The difference in performance between qbsolv and neal is also more pronounced (see Appendix B.3). *results*

We explain this performance drop by the actual composition of the datasets. While the benchmark datasets 60% and 70% of event 1004 have a similar occupancies (i.e. number of particles) and number of hits, the hit-to-noise ratio is very different: 21% for the benchmark datasets, compared to 30% and 34% for the full events (see also Section 5.1). The number of input doublets is also significantly greater (see Appendix A). *explanation*

6.6 Summary

The algorithm shows good efficiency on high p_t tracks, with a TrackML score and a recall above 90% for all tested occupancies. The purity of the results, however, is problematic: at higher occupancies, more than 50% of the final doublets are fakes. This is a shortcoming of the model itself, not of the solver used. *physics performance*

The runtime of the algorithm is distributed between model building and QUBO solving, which both rise superlinearly with the occupancy. The model building time is largely dominated by the combination of doublets into triplets and by the overhead introduced for debugging and reporting. We believe the code could be largely improved in order to achieve near linear execution times. The solving time depends on the library. Using qbsolv, it raises quickly to more than $30\,\mathrm{min}$, while using neal the run time stays below $12\,\mathrm{s}$. Qbsolv is also slower using a D-Wave, but this is mostly due to the overhead involved in using a shared, remote machine and the added cost of minor embedding. *time performance*

Playing with the sub-QUBO sizes in qbsolv or the values of the constants a and ζ in the QUBO have no significant impact on the performance. Running on full events, all the metrics drop, but this is mostly due to a higher noise-to-hit ratio. *other experiments*

7 Improvements

Chapter's content

As discussed in Section 6.3, the precision drops below 50% at around 6000 particles/event. This can be explained by the simplicity of the model. As Stimpfl-Abele already noted in the 90s,

> The biggest difference [with conventional methods] is the number of wrongly associated coordinates [...] soft constraints and very simple geometrical constants is not as good as a sophisticated algorithm based on hard constraints (fits) [24].

However, we believe the purity can be improved. This chapter first describes some properties of the fakes, then presents one solution found by the end of the project which shows promising results.

7.1 Looking at the fakes

This section looks at the results obtained using qbsolv in simulation mode on the dataset from event 1000-60% (4912 particles) using seed 13350913. The final scores for this run were $p = 60.08\%$ and $r = 93.35$.

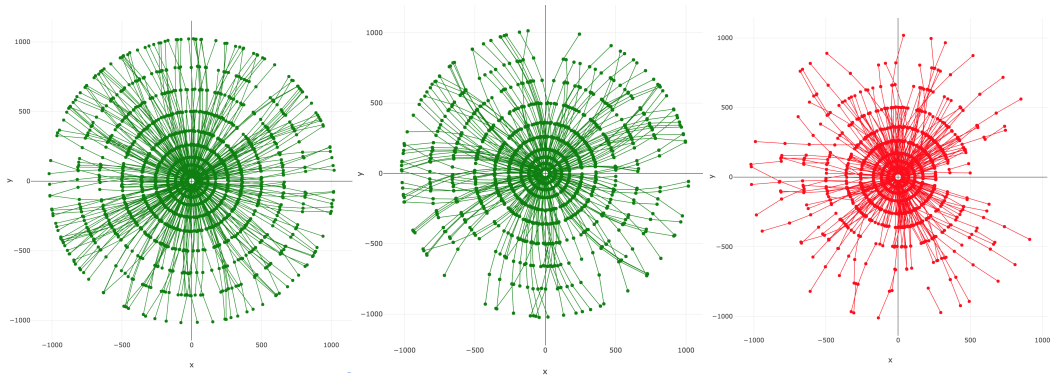
short tracks in the center

As shown in Figures 7.1 and 7.2, the fake doublets are often found in the first layers close to the center and part of very short track candidates. In this instance, track candidates with 5 hits contain 70% of the fakes, and the 90% of the fakes are in tracks of 7 hits or less. This information could be used either during model building (i.e. changing the cuts, for example the `max_path`, see Section 4.4) or during post-processing.

shape of the fakes

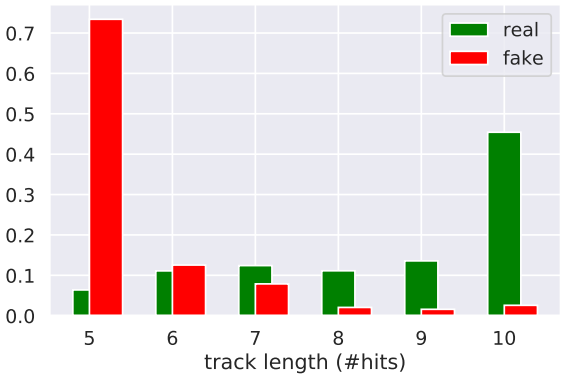
Looking at the shape of the track candidates containing fakes (Figure 7.2), they seem to be correct from our algorithm point-of-view, but many of them could be discarded by using additional physics criteria. For example, one might dismiss fakes based on:

— the magnetic field strength: some curvatures are impossible given their location in the detector;

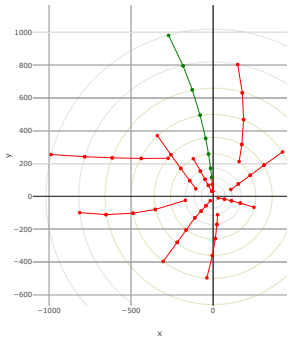


(a) Real (focused). (b) Real (unfocused). (c) Fakes.

Figure 7.1: Overview of the doublets found by type, plotted in the transverse plane.



(a) Distribution of real and fake doublets as a function of the track length it belongs to.



(b) Selection of track candidates containing fakes.

Figure 7.2: Overview of the fake doublets found.

— the origin of the vertex: neglecting multiple scattering, most vertices are found in the luminous region, so track candidates should cross the beamline at coordinates $< |55\,\mathrm{mm}|$;

— etc.

The question of whether to use those additional criteria in post-processing or in the model itself is left to discussion.

7.2 A new model

Late in the project, a discussion with other ATLAS team members raised the idea of using the impact parameter in the QUBO.

7.2.1 Overview

The current model is driven by the coupling strengths only, with a bias weight equal for all the triplets. This decision was mostly made in order to simplify the development of the first iterations. In the light of the results, using variable bias weights to discriminate between triplets

variable a_i

seems pertinent[1].

In this new model, bias weights a_i are used to discriminate between real and fake triplets by using the fact that real triplets likely point to the primary vertex whereas fake triplets do not necessarily point in any specific direction. *goal*

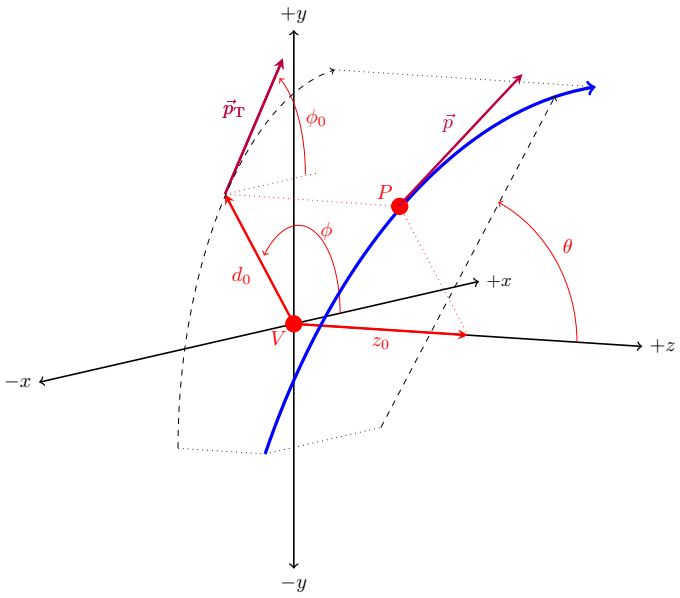


Figure 7.3: Illustration of the perigee parameters of the point of closest approach P of a track (blue curve) to the z-axis of a coordinate system centered at vertex V. The projections of the track onto the $x-y$ and $y-z$ planes are shown with the dashed black paths[35]. We are interested in the z_0, $d0$ and θ angles.

Impact parameters are widely used for this purpose in HEP experiments, specifically the transverse impact parameter, d_0, and the longitudinal impact parameter multiplied by the sine of the polar angle, $z_0 sin(\theta)$. See Figure 7.3 and the Infobox for more details. *impact parameters*

Perigee parameters

(This text is taken from github.com/ndawe/tikz-track). The helical path of a track is parametrized at the point of closest approach to the z-axis of the detector coordinate system or the z-axis of a coordinate system centered at a vertex. The *perigee parameters* $(d_0, z_0, \phi_0, \theta, \frac{q}{p})$ are illustrated below and are defined as follows:

- d_0: The transverse impact parameter is the distance of closest approach to the z-axis in the $x-y$ plane. The sign of d_0 is positive when $\phi - \phi_0 = \frac{\pi}{2} \mod 2\pi$.
- z_0: The longitudinal impact parameter is the z-coordinate at the perigee.
- ϕ_0: The azimuth angle of the track at the perigee, measured in the range $[-\pi, \pi]$.
- θ: The polar angle of the track, measured in the range $[0, \pi]$.
- $\frac{q}{p}$: The ratio of the track charge to the magnitude of the track momentum.

[1]Other experiments were made using variable weights during the project, but didn't seem to yield any improvement. One criterion used was for example the `max_path`, that is the maximal track length a triplet is potentially part of.

7.2.2 a_i computation

For each triplet T^{abc}, we determine the parameters cx, cy and cr of the circle defined by the three points (perfectly aligned points are not taken into account). Then, given (ox, oy) the coordinate of the supposed beamspot, or primary vertex of the triplet, d_0 is computed using Equation 7.1. d_0

$$d0 = \sqrt{(cx - ox)^2 + (cy - oy)^2} - cr \tag{7.1}$$

To compute z_0, we first project each doublet composing the triplet on the beamline (Z axis) and take the absolute value of the coordinate, noted z_0^{ab}. If both projections lie inside the beamspot, z_0 is set to 0. If at least one projection is too far, z_0 is computed as: z_0

$$z0 = cos(\theta_{ab})\big(max(z_0^{ab}, z_0^{bc}) - bw\big) \tag{7.2}$$

with bw the beamspot width. This is exemplified in Figure 7.4.

Finally, those two parameters are combined using: a_i

$$a_i = \alpha(1 - e^{\frac{|d_0|}{\gamma}}) + \beta(1 - e^{\frac{|z_0|}{\lambda}}) \tag{7.3}$$

The α and β parameters control the relative importance of both factors and the maximal value of the weights, while the two constants γ and λ determine at which distance (in mm) a triplet becomes really penalized.

Tracks in the TrackML dataset originate from the geometrical center, therefore impact parameters are calculated with respect to the point $(0, 0, 0)$, as opposed to the measured beamline position like for example in the ATLAS experiment [36]. Furthermore, we know from the TrackML participants document [5] that the luminous region has a width of $55\,\mathrm{mm}$. In other words, we set: *beamline parameters*

$$ox, oy = 0, 0$$
$$bw = 55$$

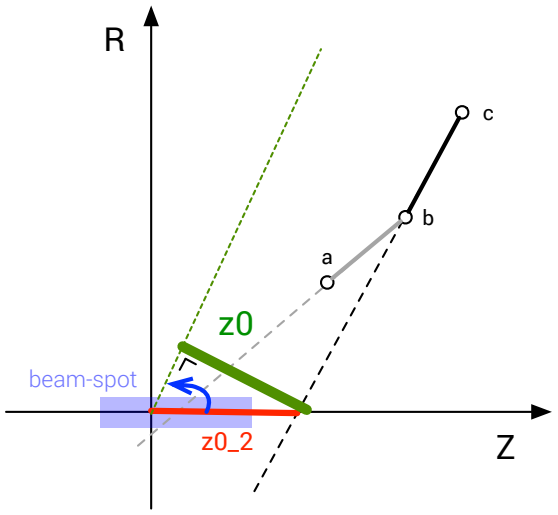


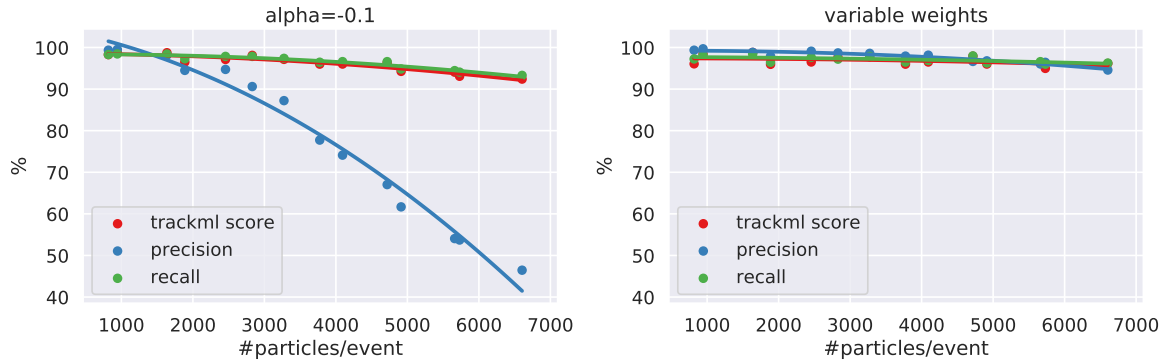
Figure 7.4: Actual z_0 used in the a_i computation.

7.2.3 Experimental results

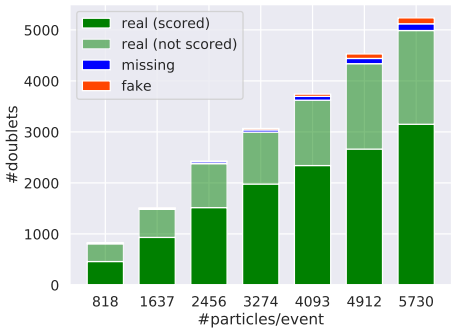
To test this new approach, the model is adapted in order to set the a_i as explained in the previous section. After some initial experiments (see Appendix B.4), the parameters of Equation 7.3 were fixed to: *parameters*

$$\alpha = 0.5, \quad \beta = 0.2$$
$$\gamma = 1.0, \quad \lambda = 0.5$$

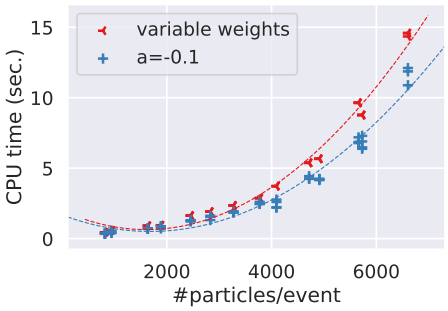
Everything else is left unchanged, and neal is used as the QUBO solver.



(a) Physics performance with (right) and without (left) variable weights.



(b) New output composition (event 1000).



(c) Timing performance using neal.

Figure 7.5: Performance of the new model.

The performance of this new model is unexpected. As shown in Figure 7.5a, the precision is now stable and stays above 94% for all tested occupancies, while the other metrics stay unchanged. With this new fake rate below 2% (see also Figure 7.5b), the model now shows a global score of 94%+ for all metrics. Moreover, this improvement comes with no significant cost on the runtime: the build time is roughly unchanged and the QUBO solving time increases by less than 0.5 s on the bigger datasets (see Figure 7.5c). *performance*

Another interesting feature is the fact that the *no conflict* constraint is better respected: upon all experiments, there was never any remaining conflict, while with the previous model we could find as much as 20 conflicts for big QUBOs. *no conflict*

To confirm those promising results, the new model was also tested on the full events discussed in Section 6.5.4. This time, there is no performance drop, letting us conclude that the new model is also less sensitive to noise. *full event*

7.2.4 Discussion

origin assumption

In track fitting, impact parameters are computed based on the reconstructed primary vertex of the track candidate. For triplets, there is no way of deriving this information and as a result the vertex is assumed to be the center of the interaction region. The reintroduction of the origin assumption can be problematic for particles originating from secondary vertices or decay, which are also of interest for the physicists.

small values

A small benchmark showed that values below 1 for γ and λ (see Equation 7.3) give better results on the current dataset (see Appendix B.4). However, this does not match the physics involved. γ and λ correspond to a cut in millimeters, and such small values would likely throw away many interesting particles in real events.

beam spread

The version of TrackML dataset used has a known bug: all particles originate from the center of the coordinate system. This might explain why the new model performs so well. Experiments on datasets with a real spread are required to really validate this new approach.

7.3 Summary

By replacing the constant linear weights in the QUBO by dynamic values derived from the triplets' impact parameters, we were able to solve the fake rate problem entirely. The new physics performance is above 94% for all metrics and seems stable as the occupancy rises. This improved model also shows a better resilience to noise and doesn't leave any conflicts in the solutions. However, it also reintroduces an origin assumption and implies some cuts that could potentially throw away interesting particles.

8 Conclusion

Chapter's content

8.1 Project summary

In this MSc thesis, we expressed the pattern recognition problem as a QUBO using a model inspired from [24]. The main novelty is the use of triplets of hits as binary variables.

dataset & experiments

Experiments were based on subsets of events from the TrackML dataset with up to ∼7000 particles and ∼43,000 hits. This dataset is in some ways simplified, e.g. it has a homogeneous solenoidal field and uses 3D space points, and the study was limited to particles in the barrel part of the detector, but it is otherwise representative of events as expected in the ATLAS detector at HL-LHC. The QUBOs were successfully solved on a D-Wave using qbsolv, and results were compared with qbsolv in classical mode and the neal library.

physics performance

Initial results showed good physics performance up to about 2000 particles/event, after which the recall stays above 90%, but the precision starts to drop. An improvement to the model introducing impact parameters in the QUBO fixed this fake rate problem. The final performance is above 94% for all metrics, with a good resilience to density and noise. Using a D-Wave showed no real advantage, yielding similar results at a much higher time cost.

timing performance

The timing performance is difficult to assess, partly because the code for building the QUBO is not optimized, QUBO solvers offer different timing performance and the D-Wave adds overhead and latencies that make comparisons difficult. Nonetheless, we believe that by improving the code and by choosing the right solvers, the runtime can be improved enough to become interesting.

8.2 Review of the objectives

This section reviews the three questions asked in the introduction.

1. Can we apply quantum annealing (QA) to the problem of particle tracking ?

Yes. This MSc Thesis expressed the pattern recognition problem as a QUBO, which is well-suited to algorithms aided by quantum annealing. This generic form is especially interesting because it is supported not only by D-Wave, but other hardware and classical libraries as well.

2. If so, is the chosen approach concretely executable on a real D-Wave ?

Yes. Experiments were successfully conducted on two D-Wave machines, a D-Wave 2000Q (latest model, available on leap) and a D-Wave 2X (Los Alamos Laboratory). However, this preliminary study shows no advantage in using a D-Wave compared to classical libraries such as neal. The necessity of splitting the problem into smaller chunks fitting the hardware, the need for minor embedding and the time overhead incurred by the shared and remote nature of the D-Wave are some of the reasons why using a D-Wave is currently suboptimal.

Even considering the potential progresses in D-Wave Systems hardware and libraries, there is still a controversy about the potential benefit of such devices. Without going into the debate, evidence shows they are able to shine, but only on QUBO problems that have very particular properties [37]. The astonishing efficiency of classical libraries on our QUBO instances makes us believe that our current problem formulation doesn't fall into this category and thus is very unlikely to ever benefit from the use of a D-Wave.

3. Is the chosen approach relevant and interesting to pursue further, and why ?

The methodology used in this study includes two parts: (A) the representation of the problem as a QUBO, and (B) the solving of the QUBO using Quantum Annealing. The physics performance obtained in this study shows that (A) is definitely worth investigating further. (B), however, is questionable. Other techniques exist to solve a QUBO that may be of greater interest, especially concerning the runtime.

An interesting side-effect of Quantum Annealing, however, is that this forced us to reformulate the tasks from a very different perspective and to think outside the box, which might lead to new ideas and a performance gain even by classical means.

8.3 Future work

There are still many improvements and possibilities left, including:

1. Code optimization. Experiments (and assessment of the approach) are mainly hindered by the time necessary to build the QUBO. Rewriting the model building algorithm to target a fast runtime is thus one of the most pressing matters.
2. Handling full events. Dealing with end-caps and double hits are challenges that need to be undertaken.
3. Handling real events. The current algorithm relies heavily on some of the simplifications of the TrackML dataset, such as the rule "at most one particle per hit". Is a QUBO still able to express the pattern recognition problem without those assumptions ?

4. Other solvers. Many other QUBO solvers are worth trying, such as the Fujitsu digital annealer [38] or the google OR-tools [39].
5. Other models. Other techniques and representations could be used in the QUBO (see Appendix D and Appendix E).
6. Model tuning. The current model has many parameters whose impact should be studied more thoroughly. This is also true for the quality cuts.

8.4 External resources & complements

The code created during this project is published on GitHub:

https://github.com/derlin/hepqpr-qallse

This MSc thesis also included an important part of scientific communication: three workshops (CPAD'18[1], ACAT'19, CTDWIT'19), a presentation at the ATLAS annual meeting, a visit to a research team in Japan, and an article soon to be published.

The presentations can be viewed using the following links:

— http://bit.ly/hepqpr-cpad
CPAD'18, *QUBO for Track Reconstruction on D-Wave* (white paper soon to be released);
— http://bit.ly/hepqpr-annual-meeting
ATLAS annual meeting, *Using a quantum annealer for particle tracking at the LHC*;

The article will be deposited on arXiv under the title "*Pattern recognition algorithms for quantum computers*" and submitted to the journal "*Computing and Software for Big Science*" [2].

8.5 Personal conclusion

This MSc thesis was an amazing experience. Being born 20 min away from CERN, it took a trip to Berkeley, CA, to learn what is really going on at the Large Hadron Collider. I was able to play with a real quantum computer, to encounter and share ideas with amazing people, to get familiar with some rudiments of particle physics and to enrich my knowledge in data science and the English language. I will be forever grateful for this opportunity.

Quantum computing is a really new and exciting technology and I wouldn't have dreamt of being part of the early adopters. Being able to access a real D-Wave and to make it execute an actual, real-purpose program is really one of the highlights of this work.

Finally, I would like to emphasize that the project itself was a real challenge that started without much expectation. Being able to achieve such good performance so quickly was unexpected and greatly appreciated. I am also really happy to have been given the opportunity to present my work in various occasions.

[1] http://www.brown.edu/Conference/CPAD2018/.
[2] https://link.springer.com/journal/41781.

A

Benchmark dataset: statistics

		#hits		#particles		input doublets			
event	%	total	noise	total	hpt	#	p	r	seed
	10	5686	1303	818	65	107316	0.45	99.79	1543636853
	20	11052	2607	1637	132	390484	0.24	99.79	1543636858
	30	16855	3910	2456	214	899983	0.17	99.36	1543636871
1000	40	22192	5214	3274	270	1515935	0.13	99.55	1543636897
	50	27807	6518	4093	332	2430144	0.10	99.01	1543636938
	60	33540	7821	4912	383	3523081	0.08	99.10	1543637005
	70	38701	9125	5730	454	4654886	0.07	99.18	1543637104
	10	6324	1310	943	87	69586	0.70	99.79	1544857310
	20	12520	2620	1887	138	267912	0.28	98.38	1544857317
	30	18805	3930	2830	228	606614	0.17	98.92	1544857331
1004	40	24844	5241	3774	290	1078231	0.13	98.68	1544857359
	50	31418	6551	4717	374	1679639	0.11	98.47	1544857402
	60	37642	7861	5661	447	2521380	0.09	98.80	1544857468
	70	43738	9172	6604	514	3315750	0.08	98.78	1544857562
	10	4741	1297	673	63	135091	0.48	99.07	1543703252
	20	9410	2595	1346	102	510608	0.20	99.14	1543703256
	30	14318	3892	2019	138	1153555	0.15	99.42	1543703265
1054	40	19238	5190	2693	192	2013481	0.10	99.24	1543703282
	50	23981	6488	3366	258	3212608	0.09	99.52	1543703311
	60	29043	7785	4039	320	4593768	0.07	99.17	1543703353
	70	33712	9083	4713	359	6109996	0.06	98.92	1543703415
1039	100	41686	12986	5471	430	4793627	0.07	99.30	1545352344
1062	100	36918	12911	4587	325	3525720	0.07	99.29	1545352499

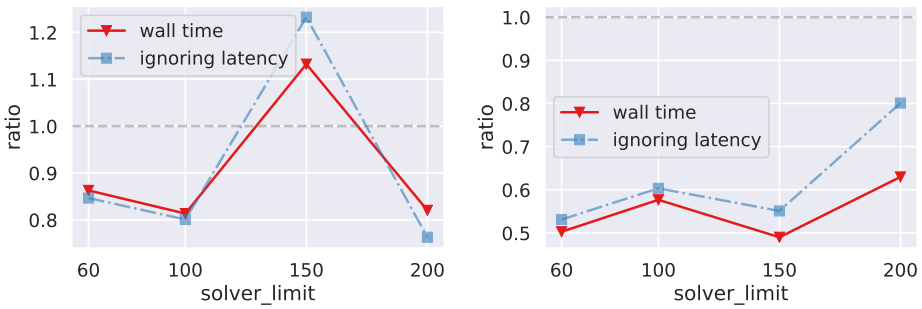
Table A.1: Benchmark datasets properties.

B

Supplementary results

Note: if not explicitly stated, those supplementary results concern the initial model and complement the results discussed in Chapter 6.

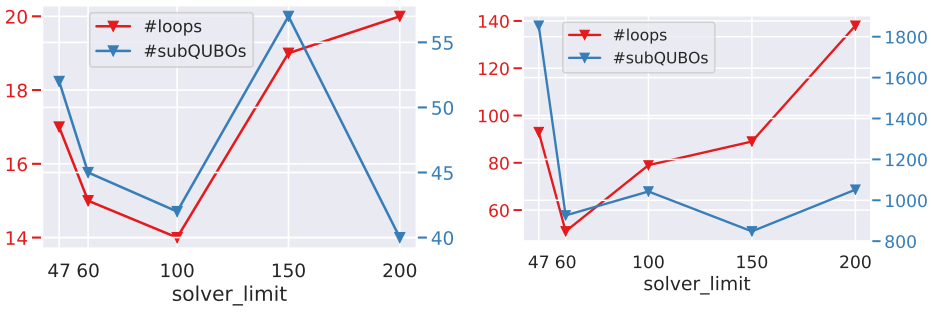
B.1 qbsolv's sub-QUBO size



(a) 10% (818 particles)

(b) 40% (3274 particles)

Figure B.1: Wall time improvement of qbsolv+D-Wave as a function of the solver limit, with or without including the internet latency. A value below 1 means a faster execution than with the default value (47).

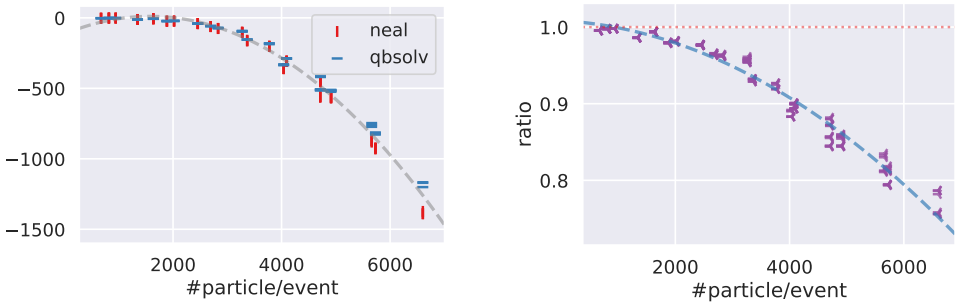


(a) 10% (818 particles)

(b) 40% (3274 particles)

Figure B.2: Number of qbsolv main loops and sub-QUBOs sampling calls as a function of solver limit.

B.2 Energy differences



(a) Difference $en_0 - en$.

(b) Ratio en/en_0.

Figure B.3: Difference between the energy of the ideal solution and the energy returned by the solvers as a function of occupancy.

B.3 Run on full events

event	model	method	precision	recall	trackml
1039	initial	qbsolv	45.00	90.91	90.85
		neal	46.50	92.57	92.54
	d0	neal	94.89	96.22	96.36
1062	initial	qbsolv	49.01	94.21	93.10
		neal	49.95	95.75	95.70
	d0	neal	96.09	96.65	96.11

Table B.1: Physics performance on full events (averaged), using the initial model (see Chapter 6) and the improved model "d0" (see Chapter 7).

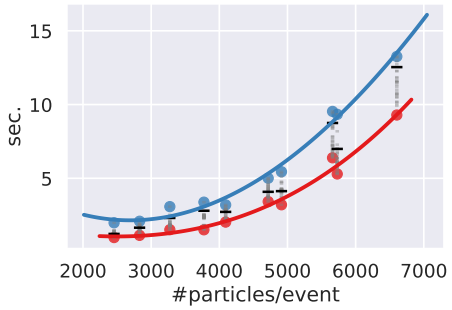
B.4 Variable weights tuning (a_i)

α	γ	β	λ	precision	recall	trackml
0.5	3	0.1	1	89.12	96.07	95.35
0.5	10	0.1	2	77.66	96.01	95.06
0.5	5	0.1	1	84.47	96.28	95.20
0.5	1	0.1	1	94.89	96.16	95.22
0.5	1	0.1	0.5	95.44	96.34	95.49
0.5	1	0	0.5	90.46	96.25	95.61
0.9	1	0.1	0.5	97.19	90.61	89.78
0.5	**1**	**0.2**	**0.5**	**96.43**	**96.34**	**95.50**
0.5	1	0.3	0.5	96.44	95.70	94.70
0.5	1	0.2	1	96.29	96.40	95.49

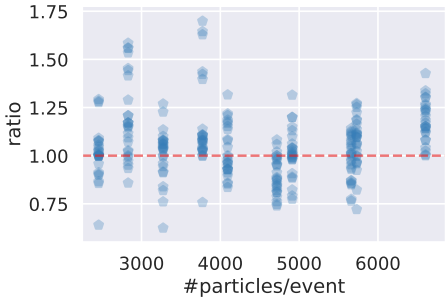
Table B.2: Improved model, results when varying the parameters in Equation 7.3 using dataset 1004-70% and neal as the QUBO solver.

B.5 QUBO parameters

Figure B.4: Physics performance for various QUBO parameters (a, ζ) as a function of occupancy.

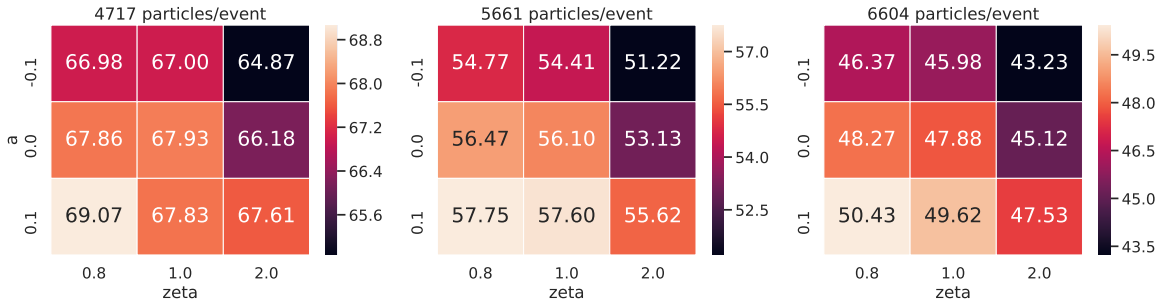


(a) Wall time as a function of occupancy. The default parameters used in benchmarks are marked in black.

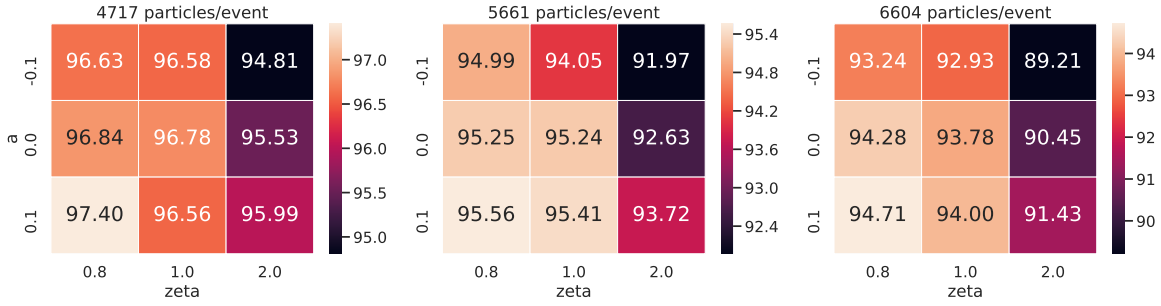


(b) Wall time improvement compared to the default parameters (ratio: default/new). A value above 1 means a smaller runtime.

Figure B.5: Wall time performance using neal and various QUBO parameters (a, ζ).



(a) Precision



(b) Recall

Figure B.6: Physics performance for various QUBO parameters (a, ζ) on a subset of occupancies.

C D-Wave timings

To have a better grasp at the D-Wave timings, we recorded the timing information available through the SAPI interface [33] during the "solver limit" experiment on dataset 1000-40% (see Section 6.5.3). Remember that for the same solver limit value, all the sub-QUBOs contain the same number of linear parameters (i.e. variables). *experiment*

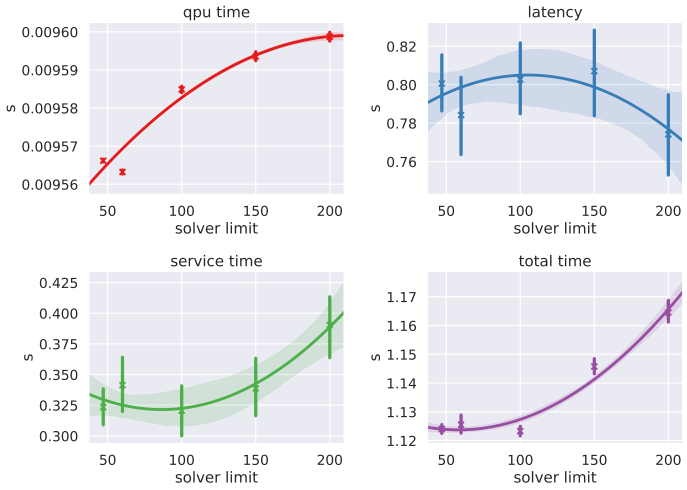


Figure C.1: Overview of D-Wave timings for sub-QUBOs of variable size (averaged).

Figure C.1 confirms that the internet latency and the service time are highly unpredictable. More surprising, the QPU access time increases with the number of sub-QUBO variables. From Figure C.2, this seems to be due to the programming time. The post-processing time is also non-negligible, but is quite variable and does not show a clear trend as the programming time does. *results*

From those experiments, one might conclude that even though D-Wave's anneal time is constant, the time required to interpret the QMI and setup the machine for its resolution is non-negligible and at least linear. The current results even show a superlinear curve, but more experiments should be undertaken to confirm this claim. *conclusion*

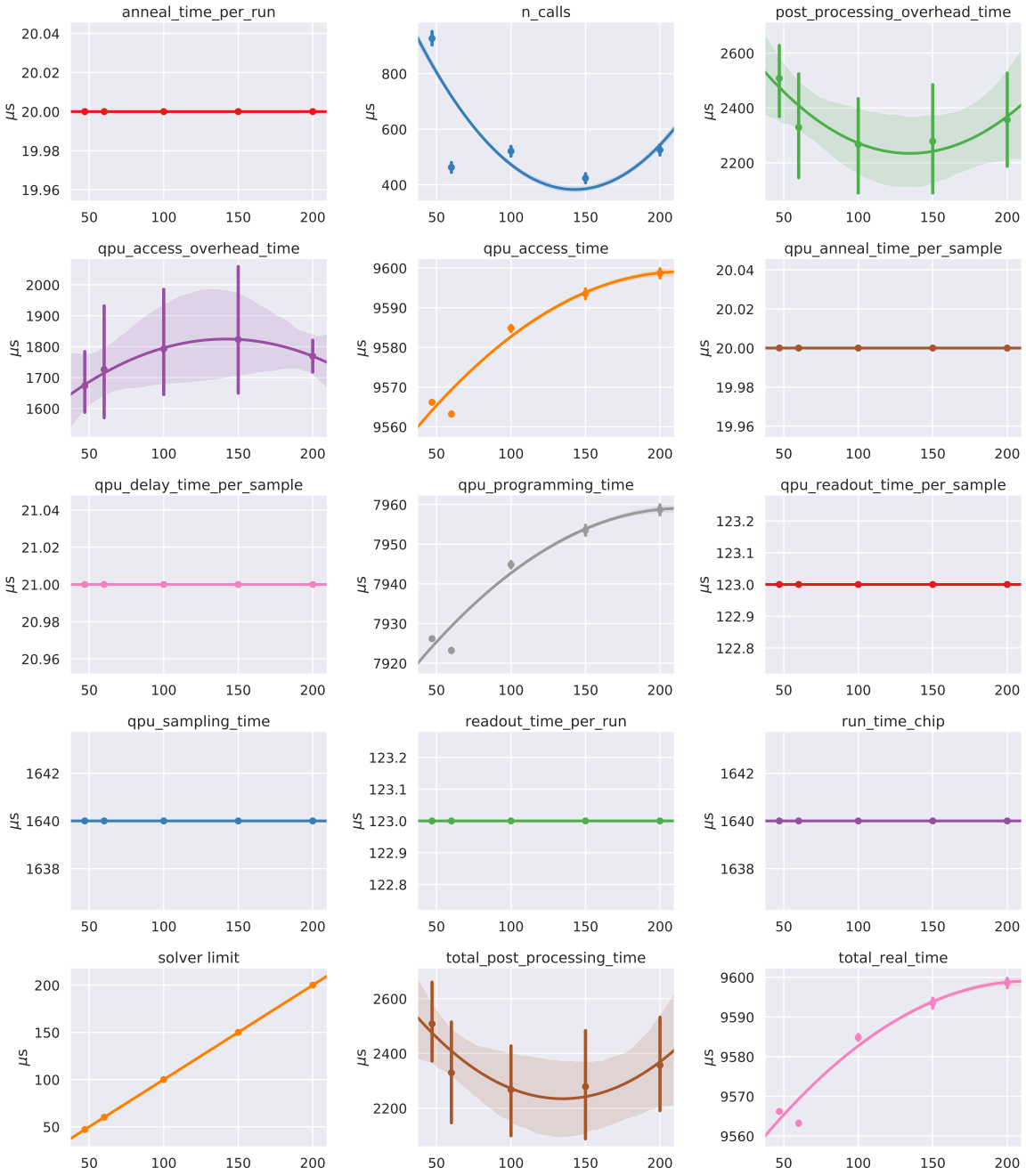


Figure C.2: Overview of all the QPU access time information available for sub-QUBOs of variable size (see [33]).

D

Alternative QUBO models

The present report discusses in length one possible QUBO modeling. To highlight that this is certainly not the only way to express track recognition as a QUBO problem, two alternatives are briefly sketched below.

D.1 A denser QUBO model

The following model relies on an idea that is central to the elastic arm approach, and is also common when formulating clustering problems as a QUBO (see e.g. [40]). We suppose to know a priori (an upper bound of) the number K of tracks that are present in the event. A special `track-0` is reserved as a container for all doublets that will not belong to any detected track.

The binary variables are every pair $< doublet, trackId >$. The Hamiltonian is composed of three parts:

$$E = E_{choose} + E_{agree} + E_{misses}$$

where:

E_{choose} formula that forces each segment to be associated to one and only one track.

E_{misses} small penalty weight for the variables where the doublet "bypasses" a layer. Maybe it is not even necessary.

E_{agree} formula that encourages or penalizes two segments to belong to the same track or not, an information that is possible to estimate between two disjoint doublets, using curvatures and R-Z angles.

It is supposed that this model would be far too dense to be used in practice on a full event. But maybe it could be an interesting alternative for some limited "disambiguation" subproblem involving O(100) hits.

D.2 A sparser QUBO model

This model computes triplets/quadruplets as usual, then defines:

— *variables*: every possible "longest path" made of consecutive quadruplets with 5 hits or more;
— *weights*: sum of the quadruplet scores in the path;
— *strengths*: repulsive constant for two paths that have a hit in common.

The number of variables would be reasonable if we manage to prevent the quadruplets from having more than one possible predecessor/successor. Then, the advantage would be the reduction of the number of couplers compared to our current model. As an example, for two 9-hit tracks that cross and share a single hit in the middle, we would need 2 variables instead of 14, and 1 coupler instead of $12 + 7$. As a solid drawback, it could be argued that most of the work is done before the QUBO solving step, which then plays a very minor role (close to simply sorting the paths according to total quadruplet score).

E

Collected ideas

It would be tedious to mention every idea that has been raised during this MSc thesis. Just to keep some of them from oblivion, below are a few directions that have been envisaged or prepared.

- ⋆ Bigger/smaller QUBOs – Use quintets and quadruplets not making the cut into the QUBO as well; drop conflict terms in the QUBO ($\sim$ *dropout* in neural networks ?) either randomly or using a clever heuristic (yet to be found).
- ⋆ Qplet model – Use quadruplets/quintets instead of triplets/quadruplets in the QUBO. This is fine for tracks of 5 hits or more.
- ⋆ Triplet's longest path – Compute for every triplet the longest path (of qplets) passing through it (this can be computed efficiently), and use it as the bias weight (or during selection). Early experiments didn't show a clear gain though.
- ⋆ Partitioning – Partition the problem by binning the hits into overlapping ϕ slices (or $\phi - \eta$ slices). This is especially interesting for high p_t datasets, where the overlap between slices can be very small.
- ⋆ QUBO for conflict resolution – Express conflicts resolution (e.g. between sub-QUBOs mentioned above) as a QUBO problem – a funny/elegant way to leverage our QUBO-centric approach and prototype.
- ⋆ Two-pass approach – Run the algorithm with a focus on high p_t tracks, remove all hits belonging to a track candidate and do a second pass with a focus on low p_t tracks. Note that this requires a good precision during the first pass.
- ⋆ Other QUBO solvers – Compare qbsolv/neal with Google OR-tools [39], Gurobi [41], MiniZink solvers [42], etc. Tools like QMASM [43] and bqp-solvers [44] could be used to convert the QUBO between the different formats.
- ⋆ Imprecision-aware feature computation, with interval-based coordinates – The detector cells correspond to an interval of (ϕ, r) coordinates which could be used to estimate worst-case vs best-case alignment scores in xplets. We wrote some preliminary pseudo-code.
- ⋆ Raw information – Use the cell signal directly instead of the reconstructed 3D coordinates (spacepoints). To combine with the previous point.
- ⋆ NumPy-centric refactoring – The current code involves a mix of different structures (objects, lists, sets, NumPy arrays). Adapting to use only NumPy arrays (or pandas dataframes) would probably boost the overall performance.
- ⋆ Magnetic field – The winner of the Kaggle challenge proved that using the magnetic field information is a powerful tool for filtering xplets.

...

F

Implementation notes

This appendix lists (a subjective selection of) some aspects which deserve a special mention. The interested reader is invited to dive into the code to see the details, or contact the author (mailto:lucy.derlin@gmail.com) for further explanations.

The code referenced here is available on GitHub (release 1.0)

https://github.com/derlin/hepqpr-qallse

Module setup There is an adequate `setup.py` file which takes care of the various dependencies, so that the installation (on Python 3.6 or 3.7) is made easy. It also defines `entrypoints`, so that the command-line scripts are available as bash commands inside the environment.

Inheritance for model variants It is often needed to try different modifications of the QUBO definition (other thresholds, strengths and weights formulas, definition of new features...) and to compare between implementations. The chosen design is based on two inheritance trees, one for the models (see Table F.1) and one for the configurations, with the use of abstract classes at the top.

Class	Description
`QallseBase`	Base class with abstract methods `is_invalid_<xplet>` and `compute_<weight>`.
`Qallse`	Default implementation, using only the 1st pass of quality cuts and constant bias weights.
`QallseMp`	Adds the 2nd pass of quality cuts. Used for benchmarks in Chapter 6.
`QallseD0`	Uses variable bias weights based on impact parameters, as described in Chapter 7.

Table F.1: Model class hierarchy (top to bottom) from the package `hepqpr.qallse`.

Visualization tools More often than not we have been very careful to design interactive vizualisation gadgets (2D and 3D) that match at best the developer's needs, for instance to highlight particular portions of fake tracks, show and hide layers, etc. This has proved very useful when trying to analyze and better understand the data and the results. See the `hepqpr.qallse.plotting` module (based on the `https://plot.ly` library).

Tracks from a set of doublets At first sight it seems a trivial task to reassemble a set of doublets into a coherent track, but it happens to be a cool algorithmic exercise to code that in both an elegant and efficient way. See the `hepqpr.qallse.track_recreater` module.

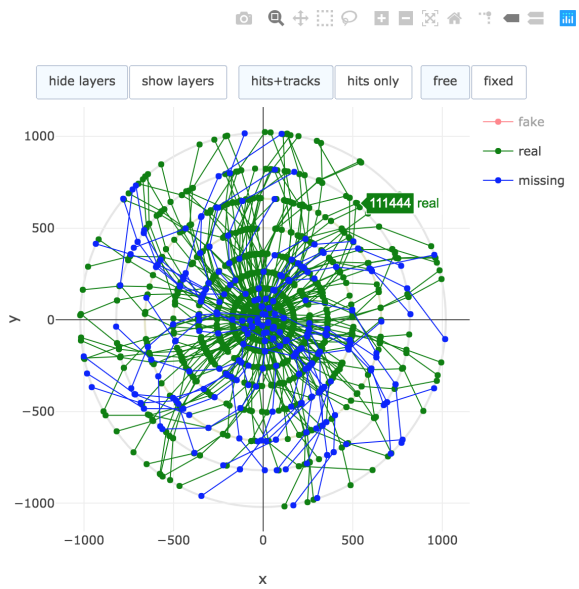


Figure F.1: Interactive visualization with zoom and selection support, toggleable legends, `hit_id` displayed on hover, png export, etc.

Conflicts within a set of triplets Two triplets can be in conflict in many different ways that might pollute the code; using Python itertools led to a very short implementation. See `hepqpr.qallse.qallse_base`, method `to_qubo`.

Parsing qbsolv verbose log The command `parse_qbsolv` creates a nice plot of the qbsolv energies after each main loop. But before the parsing itself, there is the question of stdout redirection. C shared libraries called from Python access stdout concurrently, leading to entangled outputs and broken lines. The `hepqpr.qallse.other.stdout_redirect` module contains a home-made decorator that redirects qbsolv output to a file, overcoming the limitations of existing tools like `https://github.com/minrk/wurlitzer` (not Windows-compatible + deadlocks on long lines).

Access to the detailed D-Wave timing Once we found (in the convoluted D-Wave documentation) that useful SAPI timing statistics were available from the `DWaveSampler`, the challenge was to capture it in qbsolv as well. After some digging in the code, it appeared that qbsolv exhibits a nice design that enabled us to define a thin, yet powerful, wrapper for gathering timing information. It is available in the `hepqpr.qallse.other.dw_timing_recorder` module. An example of usage is shown in Listing 3.

CLI A lot of thoughts and efforts have been put into the design of a rich and user-friendly command line interface. It relies on some advanced features of the click library to provide help, subcommands and shared options. The best example is the `qallse` entrypoint (see Listing 4), which is implemented in the package `hepqpr.qallse.cli`. This same package also defines generic functions that can be used to write benchmark scripts in minutes.

Python features Type hints and type aliases available in Python 3.6 onwards have been used extensively to make the code more robust. The `IntEnum` class was a nice and handy discovery. Python's decorators, properties and logging facilities were also extensively used, as well as my favorite feature of all: list comprehensions.

```python
from hepqpr.qallse.other.dw_timing_record import *
solver = EmbeddingComposite(DWaveSampler(config_file='path/to/dwave.conf'))
# wrap the solver and run qbsolv
# this creates one timing record for each sub-QUBO call
with solver_with_timing(solver, num_reads=10) as (wrapped_solver, records):
    response = QBSolv().sample_qubo(Q, solver=wrapped_solver)
# convert timing records dictionaries for easier manipulation
trs = [TimingRecord(r) for r in records]
# print total time and total QPU time
print('Total time', sum([t.total_time for t in trs]))
print('QPU time', sum([t.qpu_time for t in trs]))
```

Listing 3: Using a wrapper to gather D-Wave timing statistics with qbsolv.

```
Usage: qallse [OPTIONS] COMMAND [ARGS]...

  Solve the pattern recognition problem using QA.

  The <hits-path> is the path to a hit file generated using the
  `create_dataset` method. The directory should also contain a truth file
  and the initial doublets file (created either using the `-d` option of
  `create_dataset` or the `run_seeding` script).

  Output files will be saved to the given <output-path>, if any, using
  default names. If set, the <prefix> will be prepended to all output files.

Options:
  --debug / --no-debug
  -i, --hits-path TEXT         [required] Path to the hits file.
  -o, --output-path directory  Where to save the output files.
  -p, --prefix text            Prefix prepended to all output files.
  -h, --help                   Show this message and exit.

Commands:
  build       Generate the QUBO.
  neal        Sample a QUBO using neal.
  plot        Plot the final doublets and final tracks.
  qbsolv      Sample a QUBO using qbsolv (!slower!) and a D-Wave (optional).
  quickstart  Run the whole algorithm (build+neal).
```

Listing 4: CLI overview, output of the command `qallse -h` in a terminal.

Glossary

Notation	Description
3D spacepoint	Hit expressed as a 3D measurement in Euclidean coordinates (x, y, z). *Page 7, 8.*
barrel	The cylindrical-shaped layers of a detector surrounding the interaction region, in opposition to the end-caps. *Page 4, 8.*
beamline	The trajectory of the beam of accelerated particles. *Page 5, 39, 41.*
bias weight	In a D-Wave, strength of the magnetic field influencing the probability of one qubit to collapse into a given state. *Page 11, 16, 23, 39, 40.*
Chimera	The organization of the qubits inside a D-Wave processor; a lattice made of unit cells of qubits. *Page 12, 16.*
coupler	In a D-Wave, physical realization of the coupling strength; link between two qubits. *Page 11, 12, 16.*
coupling strength	In a D-Wave, how strongly the states of two qubits are intertwined. *Page 11, 16, 23, 39.*
decay	The spontaneous process of one unstable subatomic particle transforming into multiple other particles. *Page 3, 43.*
double hit	A set of hits left by the same particle on one detector layer of the detector. *Page 4, 8, 25, 36, 45.*
efficiency	Similar to *recall* in machine learning tasks, the number of correctly reconstructed trajectories divided by the number of real trajectories. *Page 26.*
end-cap	Detector in the form of disks placed at each end of a barrel-shaped detector to provide the most complete coverage in detecting particles. *Page 4, 8, 25, 26, 36, 45.*
event	The set of particle measurements (hits) in the detector associated to a single beam-beam crossing. *Page 3, 7.*
focused dataset	A dataset focusing on particles with a $p_t \geq 1\,\mathrm{GeV}$ and at least 5 hits, impacting the quality cuts applied during model creation and the computation of the performance metrics. *Page 26, 27, 29.*
ground state	The lowest-energy state of a quantum-mechanical system. *Page 10.*

Notation	Description
Hamiltonian	A mathematical description of some physical system in terms of its energies. For a quantum system, a Hamiltonian is a function that maps certain states, called eigenstates, to energies. *Page 10.*
hit	Trace left by a particle on a detector subsystem. *Page 3, 25.*
interaction region	Small region around the nominal center position of the detector (0, 0, 0) where most of the proton-proton interactions take place. *Page 3.*
luminosity	The number of collisions that can be produced in a detector per cm^2 and per second. *Page 3.*
luminous region	See interaction region. *Page 3, 41.*
minor embedding	Process of transforming a QUBO into another that fits the Chimera of a D-Wave processor, by using multiple physical qubits (*chains*) to represent one logical qubit. *Page 12, 34, 37, 45.*
missing hit	A hit that should be present, but isn't. This means a particle went through a layer of the detector undetected. *Page 4, 8, 22.*
mu (μ)	The average number of visible proton-proton interactions per bunch crossing. *Page 3, 7.*
multiple Coulomb scattering	Deviation of a particle trajectory as it passes through a material. *Page 4, 6, 8.*
noise hit	A hit that doesn't belong to any particle. This can be caused by electronic noise, random incidents, low momentum particles that don't make the cut, etc. *Page 4, 9.*
particle	The true particle trajectory inside the detector, as opposed to the reconstructed track. *Page 2.*
pattern recognition	See track finding. *Page 2, 20, 26, 45.*
pile-up	The average number of proton-proton interactions in visible events. *Page 3.*
purity	Similar to *precision* in machine learning tasks, the number of correctly reconstructed trajectories divided by the total number of reconstructed trajectories. *Page 26.*
QMI	Quantum Machine Instruction. *Page 11.*
QPU	Quantum Processing Unit. *Page 13, 16.*
QUBO	Quadratic Unconstrained Binary Optimisation. *Page 11, 16, 17, 44, 45.*
SAPI	Solver Application Programming Interface, an application layer built to provide resource discovery, permissions, and scheduling for quantum annealing resources at D-Wave. *Page 13, 16.*
track	The reconstructed trajectory of a particle. *Page 2, 20, 27.*
track candidate	A subset of hits believed to originate from the same particle; see track finding. *Page 2, 20, 31, 43.*

Notation	Description
track finding	Division of a set of hits in a tracking detector into subsets, called track candidates; each subset contains hits believed to originate from the same particle. *Page 2*.
track fitting	Aims to optimally estimate a set of track parameters from a set of hits provided by the track finding step. *Page 2*.
transverse momentum (p_t)	Component of the particle's momentum in the transverse plane, i.e. i.e. perpendicular to the beamline. *Page 3, 6*.
transverse plane	Plane perpendicular to the beamline, also called X-Y plane in detector coordinates. *Page 5*.
vertex	Production point of one or more particles. *Page 3, 20, 39, 41, 43*.
xplet	Generic term used to reference a subset of hits, encompassing doublets, triplets and quadruplets. *Page 20, 31*.

References

[1] Antonio Augusto Alves Jr et al. *A Roadmap for HEP Software and Computing R&D for the 2020s*. Tech. rep. Fermi National Accelerator Lab. (FNAL), Batavia, IL (United States), 2017.

[2] CERN - Atlas team. *ATLAS official website*. [Online; accessed 2018-09-24]. URL: https://altas.cern.

[3] CERN - Atlas team. *Charged-particle reconstruction at the energy frontier*. [Online; accessed 2018-09-24]. URL: https://atlas.cern/updates/physics-briefing/charged-particle-reconstruction-energy-frontier.

[4] Wikipedia. *pseudo-rapidity*. [Online; accessed 2018-12-25]. URL: https://en.wikipedia.org/wiki/Pseudorapidity.

[5] CERN. "Participant document: Particle Tracking and the TrackML Challenge". In: (2018).

[6] Christian Gumpert et al. "ACTS: from ATLAS software towards a common track reconstruction software". In: *Journal of Physics: Conference Series*. Vol. 898. 4. IOP Publishing. 2017, p. 042011.

[7] Kaggle. *TrackML Particle Tracking Challenge*. [Online; accessed 2018-09-24]. URL: https://www.kaggle.com/c/trackml-particle-identification.

[8] *TrackML library*. [GitHub; release V2]. URL: https://github.com/LAL/trackml-library.

[9] D-Wave Systems Inc. *D-Wave official website*. [Online; accessed 2018-09-09]. URL: https://www.dwavesys.com/.

[10] Richard Y Li et al. "Quantum annealing versus classical machine learning applied to a simplified computational biology problem". In: *NPJ quantum information* 4.1 (2018), p. 14.

[11] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. "Optimization by simulated annealing". In: *science* 220.4598 (1983), pp. 671–680.

[12] Corporate Headquarters. *Programming with D-Wave: Map Coloring Problem*. 2013.

[13] Physics World. *Quantum-computing firm opens the box*. 2011. URL: https://web.archive.org/web/20110515083848/http://physicsworld.com/cws/article/news/45960.

[14] D-Wave Systems. *What is the Chimera Topology?* URL: https://support.dwavesys.com/hc/en-us/articles/360003695354-What-is-the-Chimera-Topology-.

[15] Nike Dattani, Szilard Szalay, and Nick Chancellor. "Pegasus: The second connectivity graph for large-scale quantum annealing hardware". In: *arXiv preprint arXiv:1901.07636* (2019).

[16] D-Wave Systems. *D-Wave Systems documentation (latest)*. [Online; accessed 2019-02-01]. URL: https://docs.dwavesys.com.

[17] Jun Cai, William G Macready, and Aidan Roy. "A practical heuristic for finding graph minors". In: *arXiv preprint arXiv:1406.2741* (2014).

[18] D-Wave Systems. *GitHub organisation page*. URL: https://github.com/dwavesystems.

[19] D-Wave Systems. *Leap Cloud Service*. URL: https://cloud.dwavesys.com/leap/.

[20] D-Wave Systems. *Ocean SDK documentation (latest)*. [Online; accessed 2019-02-01]. URL: https://docs.ocean.dwavesys.com.

[21] D-Wave Systems. *qbsolv*. [GitHub; release 0.2.10]. URL: https://github.com/dwavesystems/qbsolv.

[22] M Booth, S Reinhardt, and A Roy. *Partitioning optimization problems for hybrid classical/quantum execution*. Tech. rep. dWave Technical Report. URL: https://www.dwavesys.com/sites/default/files/partitioning%5C_QUBOs%5C_for%5C_quantum%5C_acceleration-2.pdf.

[23] Fred Glover. "Future paths for integer programming and links to artificial intelligence". In: *Computers & operations research* 13.5 (1986), pp. 533–549.

[24] Georg Stimpfl-Abele and Lluis Garrido. "Fast track finding with neural networks". In: *Computer Physics Communications* 64.1 (1991), pp. 46–56.

[25] John J Hopfield. "Neural networks and physical systems with emergent collective computational abilities". In: *Proceedings of the national academy of sciences* 79.8 (1982), pp. 2554–2558.

[26] Wikipedia. *Menger curvature*. [Online; accessed 2019-01-10]. URL: https://en.wikipedia.org/wiki/Menger_curvature.

[27] Julien Esseiva. *Python seeding code*. [GitHub; commit SHA-ace405b]. URL: https://github.com/esseivaju/atlas-ml-tracking/blob/master/src/seeding.py.

[28] Ademar Tavares Delgado and Dmitry Emeliyanov. "ATLAS trigger algorithms for general purpose graphics processor units". In: *Nuclear Science Symposium, Medical Imaging Conference and Room-Temperature Semiconductor Detector Workshop (NSS/MIC/RTSD), 2016*. IEEE. 2016, pp. 1–6.

[29] JetBrains. *PyCharm*. URL: https://www.jetbrains.com/pycharm/.

[30] Project Jupyter. *Homepage*. URL: https://jupyter.org/.

[31] *CORI*. https://www.nersc.gov/users/computational-systems/cori/.

[32] vmprof. *vmprof-python*. [GitHub]. URL: https://github.com/vmprof/vmprof-python.

[33] D-Wave Systems documentation. *Timing overview*. [Online; accessed 2019-02-01]. URL: https://docs.dwavesys.com/docs/latest/c_timing_1.html.

[34] D-Wave Systems. *dwave-neal*. [GitHub; release 0.4.5]. URL: https://github.com/dwavesystems/dwave-neal.

[35] Noel Dawe. *perigee figure*. [GitHub]. URL: https://github.com/ndawe/tikz-track.

[36] ATLAS Collaboration et al. "Performance of the ATLAS track reconstruction algorithms in dense environments in LHC run 2". In: *arXiv preprint arXiv:1704.07983* (2017).

[37] Vasil S Denchev et al. "What is the computational value of finite-range tunneling?" In: *Physical Review X* 6.3 (2016), p. 031015.

[38] Fujitsu. *Digital annealer*. URL: http://www.fujitsu.com/global/digitalannealer/.

[39] Google. *About OR-Tools*. URL: https://developers.google.com/optimization/.

[40] Gary Kochenberger et al. "Clustering of microarray data via clique partitioning". In: *Journal of Combinatorial Optimization* 10.1 (2005), pp. 77–92.

[41] Gurobi. *Gurobi Optimizer*. URL: http://www.gurobi.com/products/gurobi-optimizer.

[42] MiniZinc. *Software*. URL: https://www.minizinc.org/software.html.

[43] Scott Pakin. *QMASM*. [GitHub, release v2.0]. URL: https://github.com/lanl/qmasm.

[44] Carleton Coffrin. *bqpsolvers*. [GitHub]. URL: https://github.com/lanl-ansi/bqpsolvers.