Investigating Quantum Speedup for Track Reconstruction: Classical and Quantum Computational Complexity Analysis

D. Magano[1,2], A. Kumar[1,3], M. Kālis[4], A. Locāns[4], A. Glos[5], S. Pratapsi[1,2], G. Quinta[1], M. Dimitrijevs[4], A. Rivošs[4], P. Bargassa[1,6], J. Seixas[2,7], A. Ambainis[4], and Y. Omar[1,2]

[1]Physics of Information and Quantum Technologies Group, Instituto de Telecomunicações, Portugal
[2]Instituto Superior Técnico, Universidade de Lisboa, Portugal
[3]Department of Mathematics, Clarkson University, USA
[4]Center for Quantum Computer Science, Faculty of Computing, University of Latvia, Latvia
[5]Institute of Theoretical and Applied Informatics, Polish Academy of Sciences, Poland
[6]Laboratório de Instrumentação e Física Experimental de Partículas, Portugal
[7]Center for Physics and Engineering of Advanced Materials, Portugal

22 April 2021

To investigate the fundamental nature of matter and its interactions, particles are accelerated to very high energies and collided inside detectors, producing a multitude of other particles that are scattered in all directions. As the charged particles traverse the detector, they leave signals of their passage. The problem of track reconstruction is to recover the original trajectories from these signals. This represents a very challenging data analysis problem, which will become even more demanding as the luminosity of future accelerators keeps increasing, leading to collision events with a more complex structure. Therefore, there is a pressing need to develop more efficient methods, capable of sieving through the vast amount of data produced by these machines, in order to find the extremely rare events that may contain the clues for new Physics. In light of these challenges, approaches to track reconstruction based on quantum computation have recently been put forward.

In this work, we study track reconstruction from a computational complexity perspective. Namely, we analyse a standard algorithm for tracking, the Combinatorial Track Finder, in the asymptotic limit of infinitely many recorded particles. First, we show that it is possible to improve the classical algorithm in this regime. Then, we show how to use quantum search routines to reach a lower computational complexity. And – although the found quantum speed-up is modest and of limited practical advantage in finite scenarios – to the best of our knowledge this constitutes the first theoretical proof of a quantum advantage for a state-of-the-art track reconstruction method, which we hope can further inspire classical and quantum improvements to High-Energy Physics computational challenges.

1 Introduction

Most of our understanding about the fundamental interactions and the sub-nuclear structure of matter comes from exploring the results of colliding highly energetic particles (moving close to the speed of light) in accelerator machines. These collisions produce a myriad of secondary particles, which must be detected and their trajectories subsequently reconstructed using especially designed computer algorithms. The search for new Physics beyond the Standard Model requires being able to detect and process extremely rare events among vasts amounts of data. Experimental High-Energy Physics (HEP), especially the Large Hadron Collider (LHC) programme at CERN, is for this reason one of the most demanding activities in the world in terms of computing resources [1]. This demand will grow dramatically after 2026 with the upcoming High-Luminosity phase of the LHC [2], and even more in future machines, such as the Future Circular Collider [3]. Indeed, the next generation of particle accelerators promises an increase in beam luminosity, which amounts to more produced particles and more complex collision events. With this, processing the raw data obtained in the particle detectors into useful information that can be analysed by high-energy physicists will become such a formidable task that it will likely require completely new technological paradigms. Quantum computing, promising significant speedups or reduced computational and energetic resources for specific problems, may play a key role in overcoming these challenges.

In recent years, quantum algorithms have been proposed for specific tasks in HEP data processing and analysis. In this paper, we address the problem of track reconstruction. Previous work in this direction includes solutions based on quantum annealing [4–6] and on quantum machine learning [7–11]. We note that event selection, another crucial task in HEP data processing, has also been approached using techniques from quantum annealing [12], quantum machine learning [13–16], and quantum search [17]. Complementary to this effort, [18] discusses quantum methods for the simulation of HEP events.

These promising proposals were typically conceived already with a quantum framework in mind, and were tested with very small problem instances due to the present quantum hardware limitations. Therefore, it remained an open question whether one could prove a quantum speedup for a relevant task meeting the high-volume requirements of HEP data processing. A route towards this goal is to consider the computational complexity of standard HEP algorithms and whether quantum computers could be used to improve it. In [19], the classical and quantum computational scaling of a well-known jet clustering algorithm is studied: a quantum algorithm with speedup is found, as well as an alternative classical algorithm that matches the quantum performance, therefore establishing no quantum advantage.

In our work, we analyse the computational complexity of the Combinatorial Track Finder (CTF) algorithm [20], which is the tracking[1] algorithm used by the CMS collaboration at CERN. Our analysis covers the algorithm as it is described in [20]. We note that a new version has recently been published [21], keeping essentially the same structure, with some particular improvements, as we discuss later in this paper. We show that we can reconstruct the same tracks (up to bounded-error probability) with lower quantum complexity by an adequate use of quantum search routines. While we focus on this specific tracking algorithm for concreteness, the underlying structure of the CTF, the combinatorial Kalman filter [22], is shared among several modern track reconstruction programmes [23–25]. Furthermore, the level of abstraction of our analysis facilitates extrapolating our conclusions to other situations.

Let n be the number of charged particles produced per event record. We find that the computational complexity of the CTF algorithm is $O(n^6)$ – Theorem 8. We then propose a modification to a stage of the classical algorithm that lowers the overall complexity to $O(n^4)$ –

[1]We will be using the terms "track reconstruction" and "tracking" indistinguishably.

Theorem 10. We show that we can use the quantum minimum finding algorithm [26] to speed up a subroutine of track reconstruction, reaching a quantum computational scaling of $\tilde{O}(n^{3.5})$ – Theorem 17. These results hold under the worst-case scenario where the number of reconstructed tracks scales as n^3. We also show that, depending on the number of initial track candidates formed (known as the track *seeds*), the quantum advantage may be further improved using quantum search. We also consider the case where we are given the promise that the CTF "extrapolates adequately" to the asymptotic regime, that is, the number of reconstructed tracks is $O(n)$. With this assumption, we show that we can achieve a quantum complexity of $\tilde{O}(n^3)$ using a two-level quantum minimum finding algorithm – Theorem 22.

To the best of our knowledge, this is the first theoretical proof of a quantum speedup for a relevant HEP data processing task. Of course, executing our algorithm would require access to large-scale fault-tolerant quantum computers, as well as QRAM access to the hits' data. So, we do not expect it to be applicable in the coming years. Moreover, the speedups are probably too modest to guarantee that there will be a practical advantage in using quantum computers to run the CTF algorithm (see, for example, [27]). Nevertheless, the point of our work is not that our specific proposal will necessarily be the best way to perform track reconstruction in the future. Instead, we mean to provide a new perspective on the suitability of the current tracking methods for extremely high-luminosity events, highlighting the steps of the reconstruction that could benefit from quantum processing. Ultimately, we hope that our analysis may motivate the search for more quantum solutions in this area, as we have shown that there is the possibility to improve the classical computational scaling.

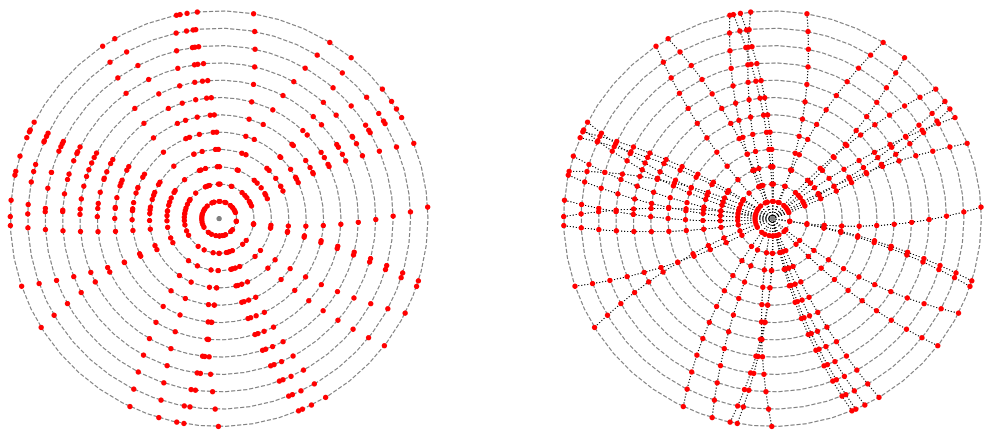
1.1 Structure of the Paper

We organize this paper as follows. In Section 2 we summarize some preliminary information that will be useful for understanding the subsequent sections of the paper. We start by providing a qualitative overview of the problem of track reconstruction (Section 2.1), following with a brief description of the computational models underlying our analysis (Section 2.2). In Section 3 we introduce our model of tracking and describe the CTF algorithm (Sections 3.1-3.4). The classical complexity of the CTF is stated in Section 3.5. Then, we propose a classical modification to the CTF algorithm in Section 4. It is in Section 5 that we introduce quantum algorithms for track reconstruction. We start by describing how to perform seed generation with quantum search in Section 5.1. In Section 5.2 we show how a routine based on quantum minimum finding permits reaching a lower complexity than the classical CTF even in the worst-case scenario. Finally, in Section 5.3 we propose a two-level quantum minimum finding algorithm that solves tracking with an improved scaling depending on the total number of reconstructed tracks. Before concluding (Section 7), we devote Section 6 to discussing the assumptions of our analysis.

2 Preliminaries

2.1 Overview of Track Reconstruction

In particle physics experiments, particles are accelerated to very high energies and collide in bunches inside tracking detectors. In these collisions, new particles are created and scattered in all directions. As charged particles cross the detector's multiple layers, they leave signals of their passage, which are converted into three-dimensional points called *hits*. The collection of hits that was left by such a particle is called that particle's *track*. The event record of an experiment consists of the totality of signals from all particles of an interaction (or possibly several interactions) after one full readout of the detector. The goal of tracking is to reconstruct



(a) The input to tracking is a set of hits (red circles) corresponding to detections of the particles' passage.

(b) We recover the original trajectories (dotted black lines) by grouping hits that belong to the same particle.

Figure 1: Illustration of track reconstruction. Transverse view of a tracking detector with cylindrical layers (dashed grey lines).

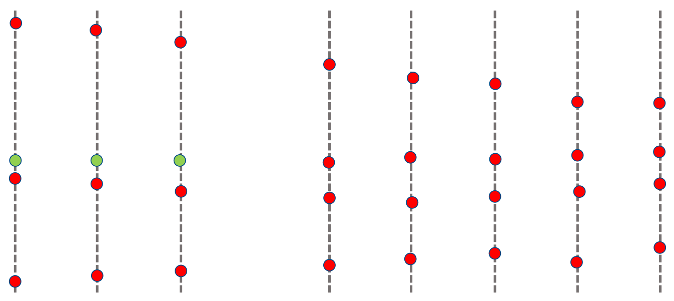
the particles' tracks from the event record. See Figure 1 for an illustration. We can use dynamical models for the particles' trajectories to discriminate which hits belong to the same particle track. However, given that in real experiments an event can contain several thousand hits, most of the combination of hits (candidate tracks) will not correspond to an actual particle. So, we need efficient algorithms to reconstruct the tracks in useful time.

Different approaches in local and global methods of tracking have been applied in experimental particle physics [28], each with its strengths and shortcomings. The quantum proposals so far [4–10] are based on global methods, treating all hit information in an equal and unbiased way. In contrast, here we consider the reconstruction algorithm used at the CMS experiment, known as the Combinatorial Track Finder (CTF) [20], which belongs to the category of local methods. Like all algorithms of this type, it involves three main ingredients. First, it requires a method to generate track *seeds*, which consist of rudimentary initial track candidates formed by just a minimal set of hits. Then, it relies on a *parametric track model*, which assigns to a track a set of trajectory parameters and provides a method of extrapolation along the trajectory. Finally, a *quality criterion* distinguishes good track candidates from the fake ones, discarding the latter from the solution. In Figures 2 and 3 we sketch the CTF algorithm. In Section 3, we explain it in detail and study its computational complexity.

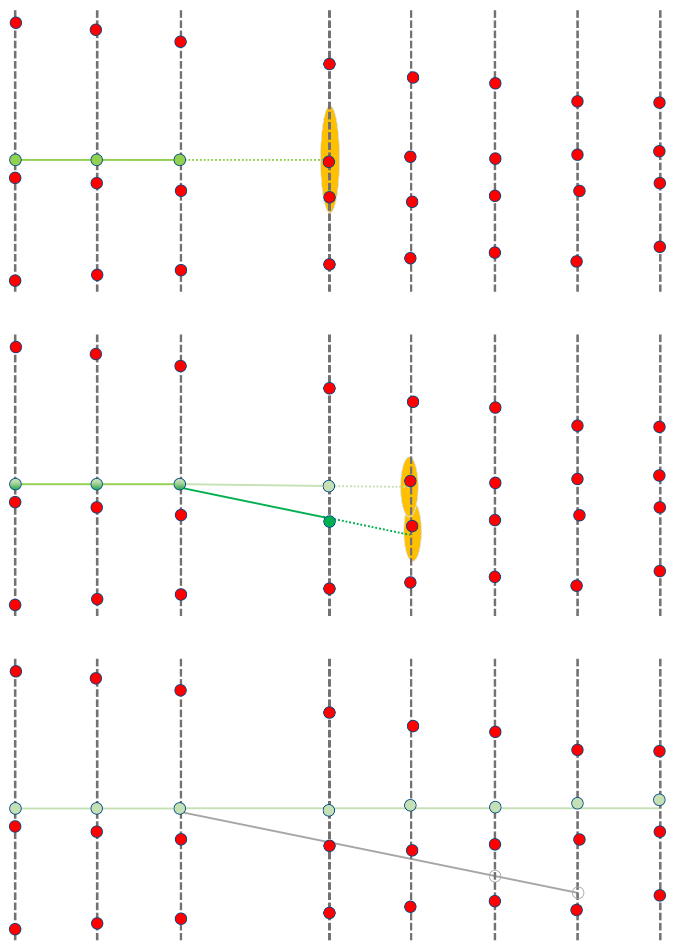
2.2 Computational Model

The running time of the CTF algorithm is the result of different factors. Naturally, the larger the number of recorded particles the more demanding track reconstruction becomes. At the LHC, sophisticated computational architectures are employed to optimize the running time. Furthermore, the parameters of the tracking software are carefully adjusted to achieve in useful time a track reconstruction with the desired accuracy, under some assumptions on the observed events.

In this work, we offer a different perspective on tracking, focusing on the computational complexity of the problem. In other words, we are interested in understanding how it fundamentally

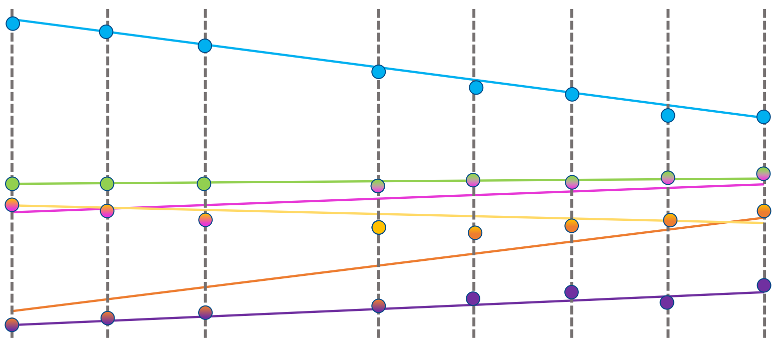


(a) Seed Generation. In the CTF algorithm we start by forming seeds, that is, triplets of hits from the three inner layers. We highlight in green a possible seed.

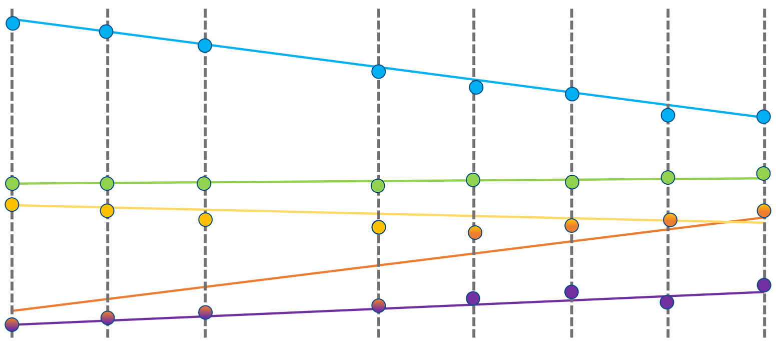


(b) Track Finding. The trajectory defined by the seed highlighted in panel 2a is continued until it meets the next layer. If we find more than one hit close to the predicted trajectory, we may form new tracks that are then continued independently. This procedure is repeated until we reach the end of the detector – the track is accepted to the next stage – or we find too many layers without a close hit – the track is dropped.

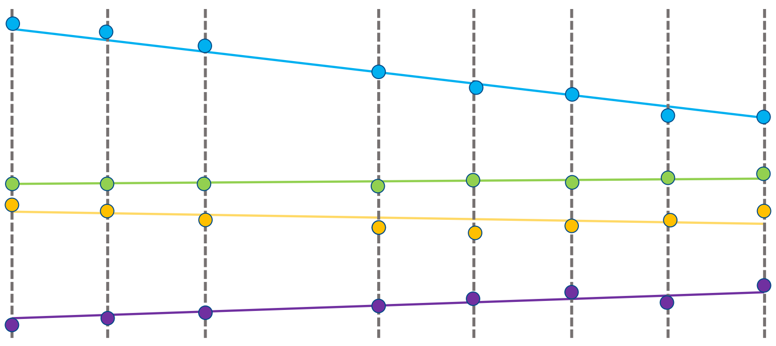
Figure 2: The first two stages of the Combinatorial Track Finder (CFT) algorithm: seed generation and track finding. The charged particles resulting from a collision travel from left to right, their hits (red dots) behind recorded as they cross the detector layers (dotted grey lines, the leftmost ones corresponding to the inner layers).



(a) Candidate Tracks. Suppose that the track finding stage outputs these tracks. Different colours correspond to different tracks (hits in more than one track have mixed colours).



(b) Track Cleaning. The candidate tracks are submitted to a cleaning stage. If two tracks share too many hits, one of them gets discarded. This is what happens to the pink track in panel 3a.



(c) Track selection. A final selection stage based on the quality of fit of the trajectories to the hits permits eliminating some fake tracks (like the orange one in panels 3a and 3b). This final panel represents the output of the track reconstruction.

Figure 3: The following stages of the Combinatorial Track Finder algorithm (continued from Figure 2): track cleaning and track selection.

scales with input size. As is common in the theoretical analysis of algorithms, we will concern ourselves with the asymptotic limit of arbitrarily large number of particles. We adopt the standard "big O" notation for asymptotic upper bounds. For two functions f and g from $\mathbb{N}$ to $\mathbb{R}$ we say that $f = O(g)$ if $\exists c, n_0 > 0 : \forall n, (n > n_0 \implies f(n) < c \cdot g(n))$. We write $f = \Omega(g)$ if $g = O(f)$. We say that $f = \Theta(g)$ if $f = O(g)$ and $g = O(f)$. By "constant time", we mean $O(1)$.

For the analysis of the classical algorithms, we assume that we can access the coordinates of a given hit in constant time. Moreover, simple arithmetic operations on the hit's coordinates are counted as taking $O(1)$ time. In the context of the quantum algorithms we work in the circuit model, measuring time as the number of quantum gates used. We further assume access to a QRAM that is able to load classical data in coherent superposition in logarithmic time in the number of memory cells [29, 30]. These choices represent the distinct standard practices in classical/quantum algorithm analysis. To attenuate the differences in the computational models, we present the results in $\tilde{O}$ notation, that is, omitting the poly-logarithmic dependencies in the complexities (for example, we say that $\sqrt{n}\log^3 n = \tilde{O}(\sqrt{n})$).

Another difference between the classical and quantum scenarios is that all the presented classical algorithms are deterministic, meaning that for a given input they will always output the same answer. On the other hand, our quantum algorithms are probabilistic. That is, for a given input, they output the correct answer with probability at least $1 - \epsilon$, where ϵ is some positive constant.

3 The Combinatorial Track Finder

We present a simplified model of tracking. In what follows, we will omit several details that are crucial in practice, but that do not significantly influence the complexity analysis. In Section 6, we discuss some of our assumptions and the validity of our model.

Assumptions about the particles' trajectories. Let n denote the number of charged particles present in the event record. Their trajectories originate from a fixed interaction region, but we do not assume that they come from a single collision spot. The detector is immersed in a quasi-uniform co-axial magnetic field, so we expect the particles to follow helical trajectories aligned with the field's direction. These trajectories can be described by five parameters [31]. Let $\mathcal{P} \subset \mathbb{R}^5$ be the five-dimensional cuboid corresponding to the trajectories' parameter space. Each experiment that produces a sequence of particle trajectories is governed by a physical process that implicitly selects these parameters for each trajectory; in the forthcoming, we call this procedure an *event*. We model this formally as a random variable $\pi : \Omega \to \mathcal{P}$ that selects a parameter with respect to a probability space $(\Omega, \mathcal{F}, P)$ that accounts for various physical parameters, such as noise, etc.. We make the mild assumption that π follows a probability distribution p_π on $\mathcal{P}$ that is strictly positive. With our model, an event is generated by drawing n random samples $\pi_1, \ldots, \pi_n$, each following p_π (that is, we treat them as i.i.d. random variables).

Assumptions about the detector's layers. The detector has a fixed geometry with a discrete set of sensor layers. We consider that the detector has L cylindrical layers aligned with the beam line. The layers are indexed from 0 to $L - 1$ (the most inner layers having the smallest indices). We assume that each particle traverses every layer and that they never return to a previously visited layer. We assume the layers to be continuous two-dimensional surfaces $\mathcal{C}_l$, $l \in \{0, \ldots, L-1\}$.

Assumptions about the hits' data. As a particle traverses a layer, it leaves a complex signal resulting from the interaction with the sensor's pixels. We treat these signals simply as three-dimensional points, called hits. We assume that the granularity of the sensors is high enough such that each detected hit can be differentiated from other hits. Since each trajectory leaves a unique hit on each layer l, formally we have a continuous map $H_l : \mathcal{P} \to \mathcal{C}_l$, relating each trajectory to its point of intersection with layer l. At every layer, we identify the hits by labels from $\{0, \dots, n-1\}$. We use the notation $\mathbf{m}_{l,j}$ for the coordinates of jth hit in layer l. It is possible that some hits are not measured at all due to sensor inefficiencies. That is, we do not necessarily have a hit $\mathbf{m}_{l,j}$ for every pair (l, j). This means that we may not be able to tell the exact value of n directly from the event record, as it is possible that there is no layer registering all particles. In that case, we would be indexing the hits with labels from $\{0, \dots, n^* - 1\}$, where n^* is the largest number of hits measured in any layer. We consider that $n^* = n$ for simplicity, but every result in this paper would hold the same as long as $n^* = \Theta(n)$.

Under these assumptions, we expect the number of measured hits per layer to grow proportionally to n. Moreover, we can establish the following useful lemma:

Lemma 1. *For almost all events, given any fixed, open (non-empty) subset of any detector layer, the number of hits in that subset grows as $\Theta(n)$.*

Proof. The trajectories are determined by the parameters in $\mathcal{P}$. If $S \subset \mathcal{C}_l$ is open and non-empty, then $U_S := H_l^{-1}(S) \in \mathcal{P}$ is open and non-empty and by assumption, the probability that a trajectory has parameters in U_S is $p_S := p_\pi(U_S) > 0$. If $\pi_1, \dots, \pi_n$ are n random samples of parameters drawn without replacement and following p_π, then the strong law of large numbers implies that almost surely, the number of parameters sampled from U_S grows as $\Theta(n)$. Applying H_l to this gives the growth of the number of hits in S. $\square$

For our complexity analysis we only consider the dependence on the variable n. Evidently, the data in the event record also depends on quantities like the number of layers of the detector, the granularity of the sensors, or the efficiency of the detectors. But these are fixed from the experimental hardware and do not vary from event to event. On the other hand, we expect the average n to grow as we increase the beam's instantaneous luminosity. Thus, we believe n to be an appropriate measure of the size of the input to the tracking problem. This simplification, relying on some level of abstraction from the actual experiment, allows us to draw some conclusions on the computational complexity of tracking, in particular on the complexity of the CTF algorithm.

We separate the analysis of the CTF algorithm into four stages[2]: seed generation (Section 3.1), track finding (Section 3.2), track cleaning (Section 3.3), and track selection (Section 3.4). These stages are executed sequentially, as each depends on the output of the previous one. Thus, each of them contributes additively to the overall complexity.

3.1 Seed Generation

The purpose of this stage is to provide initial track candidates, formed by three hits, and their trajectory parameters. The CTF algorithm generates seeds by choosing triplets of hits from the three most inner layers – see Algorithm 1 for a pseudocode representation. Recall that we are trying to fit the tracks into helices aligned with the detector axis. So, we cannot fix the five parameters defining such helices with fewer than three points. Rigorously, in general three points determine a countable family of such helices. If we assume that the trajectories do not realize

[2]For the readers familiar with [20], we warn that our division of the CTF algorithm into four stages is not the same as in the original paper. We proceed this way for convenience of the complexity analysis.

"a full turn" between these points, this degeneracy is broken. However, then we do not have the guarantee that there is an helix passing exactly through the three points. In practice, as there are experimental uncertainties about the hits' positions, this is not a concern.

Algorithm 1: Seed Generation

```
  input  : event record
  output: seeds
1 foreach hit $\mathbf{m}_{0,j_0}$ in layer 0 do
2     foreach hit $\mathbf{m}_{1,j_1}$ in layer 1 do
3         foreach hit $\mathbf{m}_{2,j_2}$ in layer 2 do
4             seed ← $(\mathbf{m}_{0,j_0}, \mathbf{m}_{1,j_1}, \mathbf{m}_{2,j_2})$;
5             if seed is good then
6                 add seed to output;
```

Because there are $O(n)$ hits per layer, we immediately see that:

Theorem 2. *Seed Generation (Algorithm 1) has complexity $O(n^3)$.*

To limit the number of formed seeds, we require them to satisfy certain restrictions. Namely, we require a minimum transverse momentum (which is proportional to the radius of the projected trajectory onto the transversal plane) and maximum transverse and longitudinal distances of the point of closest approach to the presumed beam-spot. Nevertheless, these constraints do not change the complexity. Indeed, each pair of hits in the first two layers determines an area in the third layer where an acceptable seed with these hits could be found – see Figure 4. According to Lemma 1, in the worst-case scenario for any fixed area in a sensor we may have $\Theta(n)$ hits. So, we have to admit that we may form $O(n^2 \cdot n)$ seeds. In summary,

Lemma 3. *In the worst-case scenario there are $\Omega(n^3)$ good seeds (that is, Algorithm 1 is optimal for seed generation).*

3.2 Track Finding

The track finding stage extrapolates the seeds' trajectories along the expected path of the particle and builds track candidates by adding compatible hits from successive detector layers, updating the parameters at each layer (see Figure 2b). More precisely, the track finding strategy is based on the combinatorial Kalman filter [22], which is an adaptation of the Kalman filter [32] for tracking problems. We now describe this method.

Say that a trajectory at layer $l-1$ is described by a five vector $\mathbf{p}_{l-1}$. The propagated state vector $\mathbf{p}_l$ at next layer is modelled by the system equation

$$\mathbf{p}_l = \mathbf{F}_l \mathbf{p}_{l-1} + \mathbf{w}_l. \tag{1}$$

$\mathbf{F}_l$, known as the process matrix, describes the propagation of a charged particle in a uniform magnetic field from layer $l-1$ to l. $\mathbf{w}_l$ is a random variable called process noise. A measurement $\mathbf{m}_l$ at layer l is given by

$$\mathbf{m}_l = \mathbf{H}_l \mathbf{p}_{l-1} + \mathbf{e}_l, \tag{2}$$

where $\mathbf{H}_l$ is the measurement matrix and $\mathbf{e}_l$ is the measurement noise. We assume that we know the covariance matrices for the process and measurement noises. Note that, in general, we could

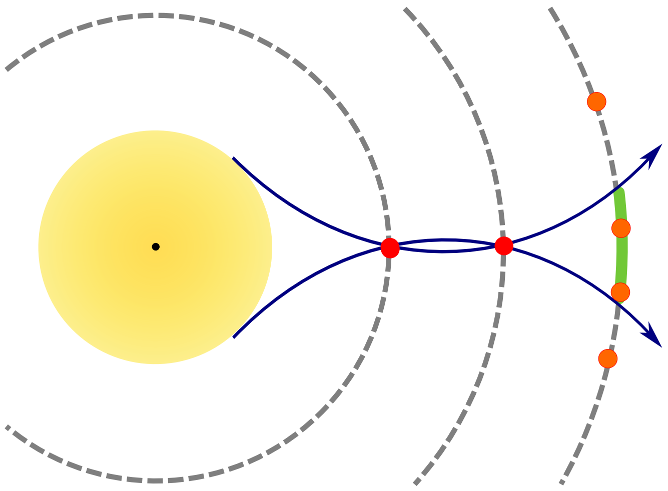


Figure 4: Seed Generation. The dotted grey lines represent the first three detector layers (transversal view). The yellow region around the beam axis (the black dot) is the region where we admit the collisions may occur. In this illustration, we are forming a seed with the two hits from the inner layers marked in red. For any hit in the third layer (orange circles) to be included in this seed, it must belong the green region. This is the region where the trajectories that respect the seeding criteria and that pass through the two red hits cross. In blue we draw two trajectories originating from the outer edge of the collision region.

replace equations (1) and (2) by non-linear relations. But the linear model usually suits the purpose of track reconstruction.

Suppose that we have built a track up to layer $l-1$ with the measurements (i.e., hits) $\mathbf{m}_{0,j_0}, \mathbf{m}_{1,j_1}, \ldots, \mathbf{m}_{l-1,j_{l-1}}$. With this information, we describe our prediction of the trajectory at this layer by a state vector $\mathbf{p}_{l-1|l-1}$ and corresponding covariance matrix. Without knowing which hit from layer l belongs to this track, we predict that the state vector at layer l is

$$\mathbf{p}_{l|l-1} = \mathbf{F}_l \mathbf{p}_{l-1|l-1}. \tag{3}$$

We say that the predicted measurement at layer l is

$$\mathbf{m}_{l|l-1} = \mathbf{H}_l \mathbf{p}_{l|l-1}. \tag{4}$$

This would be the location of the lth hit if we had perfect knowledge of the trajectory and there were no process/measurement errors. In reality, we do not expect to find any hit exactly at this predicted measurement. When considering an actual measurement $\mathbf{m}_{l,j}$, we say that the predicted residual is

$$\mathbf{r}_{l|l-1}\left(\mathbf{m}_{l,j}\right) = \mathbf{m}_{l,j} - \mathbf{m}_{l|l-1} \tag{5}$$

The predicted χ^2 value is defined as

$$\chi^2_{l|l-1}\left(\mathbf{m}_{l,j}\right) = \mathbf{r}_{l|l-1}\left(\mathbf{m}_{l,j}\right)^T \mathbf{R}^{-1}_{l|l-1} \mathbf{r}_{l|l-1}\left(\mathbf{m}_{l,j}\right), \tag{6}$$

where $\mathbf{R}_{l|l-1}$ is the covariance matrix of the predicted residual. Intuitively, a high χ^2 value tells us that the measurement is unlikely to belong to the track. Then, when evaluating which hit to add to the track, only the ones whose predicted χ^2 value is below some fixed threshold χ^2_0 pass to the filtering phase. Suppose that $\mathbf{m}_{l,j}$ satisfies this criterion. Based on this measurement, we update the state vector prediction to

$$\mathbf{p}_{l|l} = \mathbf{p}_{l|l-1} + \mathbf{K}_l \mathbf{r}_{l|l-1}\left(\mathbf{m}_{l,j}\right), \tag{7}$$

where $\mathbf{K}_l$ is the Kalman gain matrix, which is calculated based on the covariance matrices of state vector, the process noise and the measurement noise (see [22] for explicit expression). We say that the filtered residual for this measurement is

$$\mathbf{r}_{l|l}\left(\mathbf{m}_{l,j}\right) = \mathbf{m}_{l,j} - \mathbf{H}_l \mathbf{p}_{l|l}. \tag{8}$$

The filtered χ^2 value is

$$\chi^2_{l|l}\left(\mathbf{m}_{l,j}\right) = \mathbf{r}_{l|l}\left(\mathbf{m}_{l,j}\right)^T \mathbf{R}_{l|l}^{-1} \mathbf{r}_{l|l}\left(\mathbf{m}_{l,j}\right), \tag{9}$$

$\mathbf{R}_{l|l}$ being the covariance matrix of the filtered residual. One can show that the predicted and filtered χ^2 values are actually identical (see [22]), that is,

$$\chi^2_{l|l-1}\left(\mathbf{m}_l\right) = \chi^2_{l|l}\left(\mathbf{m}_l\right), \forall \mathbf{m}_l \in \mathbb{R}^3 . \tag{10}$$

This means that we can determine the filtered χ^2 value without explicitly updating the trajectory. The total χ^2 value of the track at layer l is the sum of the filtered (or predicted) χ^2 values from all previously visited layers

$$\chi^2_{\leq l}(\mathbf{m}_{0,j_0}, \dots, \mathbf{m}_{l,j_l}) = \sum_{i=0}^{l} \chi^2_{i|i}(\mathbf{m}_{i,j_i}). \tag{11}$$

In general, we may have several hits passing to the filtering phase. As we are not sure which one truly belongs to the track, we form new candidate tracks each including a different hit. These tracks are then followed independently. Also, to accommodate the possibility of detection inefficiencies the CTF permits adding a "ghost hit" if no suitable hit is found. However, to avoid a rapid increase in the number of tracks, we impose a limit of λ tracks retained at each step (the default in [20] being $\lambda = 5$). If at any point this limit is surpassed, we abandon the worst tracks. To decide this, each track candidate is attributed a quality score q_l of the form

$$q_l = l - m_{ghost} - \omega \cdot \chi^2_{\leq l}, \tag{12}$$

where m_{ghost} is the number of ghost hits included in the track and ω is some configurable weight (we omitted the dependence on the measurements). At any step we can discard a candidate track if it contains too many ghost hits or the total χ^2 value exceeds a given threshold. Otherwise, the procedure is continued until the end of the detector is reached (that is, we arrive at $l = L - 1$). The quality score (12) at that point is said to be the quality score of the track candidate. The tracks that reach this step are accepted for the next stage of the CTF algorithm. For a summary description of track finding see Algorithm 2.

Before evaluating the complexity of the finding stage, it is important to understand how many candidate hits pass to the filtering phase. The space of points with acceptable predicted χ^2 value (equation (6))

$$\{\mathbf{m_l} \in \mathbb{R}^3 : \chi^2_{l|l-1}\left(\mathbf{m}_l\right) < \chi^2_0\} \tag{13}$$

is an ellipsoid around the predicted measurement. The intersection of this ellipsoid with the layer's surface yields a region in that layer whose area is independent of n. By Lemma 1, we may find $\Theta(n)$ hits in that region. Therefore,

Algorithm 2: Track Finding

```
input  : seeds, generated by Algorithm 1; event record
output: candidate tracks
1  foreach seed do
2      initialize empty list candidate_tracks;
3      estimate initial state vector $\mathbf{p}_{2|2}$ and quality factor $q_2$ for seed;
4      add (seed, $\mathbf{p}_{2|2}$, $q_2$) to candidate_tracks;
5      foreach layer $l$ from 3 to $L-1$ do
6          foreach (track, $\mathbf{p}_{l-1|l-1}$, $q_{l-1}$) in candidate_tracks do
7              evaluate predicted measurement $\mathbf{m}_{l|l-1}$;
8              foreach hit $\mathbf{m}_{l,j}$ in layer $l$ do
9                  if $\chi^2_{l|l-1}(\mathbf{m}_{l,j}) < \chi^2_0$ then
10                     new_track ← track + $\mathbf{m}_{l,j}$;
11                     form new candidate track for seed with $\mathbf{m}_{l,j}$;
12                     evaluate $\mathbf{p}_{l|l}$ and quality factor $q_l$ for new_track;
13                     add (new_track, $\mathbf{p}_{l|l}$, $q_l$) to candidate_tracks;
14             if there is no hit $\mathbf{m}_{l,j}$ in layer $l$ such that $\chi^2_{l|l-1}(\mathbf{m}_{l,j}) < \chi^2_0$ then
15                 new_track ← track + $\mathbf{m}_{l|l-1}$;                    /* ghost hit */
16                 evaluate $\mathbf{p}_{l|l}$ and quality factor $q_l$ for new_track;
17                 add (new_track, $\mathbf{p}_{l|l}$, $q_l$) to candidate_tracks;
18             remove (track, $\mathbf{p}_{l-1|l-1}$, $q_{l-1}$) from candidate_tracks;
19         select the best $\lambda$ tracks of candidate_tracks ;          /* sorting by $q_l$ */
20     add elements of candidate_tracks to output;
```

Lemma 4. *The filtering step takes $O(n)$ time.*

Starting from a single seed, we only propagate up to $\lambda = O(1)$ tracks from layer to layer. For each of these, the analytical continuation of the trajectory from one layer to another (equations (3) and (4)) is performed in $O(1)$ time. As we have seen with Lemma 4, performing filtering requires $O(n)$ time per track candidate. Finally, in $O(n)$ time we can determine the λ tracks with best quality score (12) that are propagated to the next layer. The number of layers L is $O(1)$. Thus, we arrive at

Theorem 5. *Track Finding (Algorithm 2) has complexity $O(k_{seed} \cdot n)$, where k_{seed} is the number of seeds that were formed in seed generation stage.*

3.3 Track Cleaning

With the method described so far we may find multiple tracks corresponding to the same particle, either starting from different seeds, or when a given seed develops into more than one track. To avoid this, the cleaning stage calculates the fraction of shared hits between all pairs of track candidates

$$\frac{N^{hits}_{shared}}{\min\left(N^{hits}_1, N^{hits}_2\right)}, \tag{14}$$

where N^{hits}_1 (N^{hits}_2) is the number of hits used in forming the first (second) track and N^{hits}_{shared} is the number shared hits between the two tracks. If for any pair this fraction exceeds a fixed threshold value, the worst track (i.e., the one with lowest quality score) gets discarded. This is represented in Algorithm 3.

Algorithm 3: Track Cleaning

```
  input : candidate tracks, generated by Algorithm 2
  output: cleaned candidate tracks
1 foreach $\mathsf{track}_1$ in candidate tracks do
2     foreach $\mathsf{track}_2$ (different from $\mathsf{track}_1$) in candidate tracks do
3         if $\mathsf{track}_1$ and $\mathsf{track}_2$ share more than allowed fraction of hits then
4             remove the one with lowest quality score from the set of candidate tracks;
5 output remaining candidate tracks;
```

We see that

Theorem 6. *Track Cleaning (Algorithm 3) has complexity $O(k^2_{find})$, where k_{find} is the number of track candidates that were formed in the track finding stage.*

3.4 Track Selection

For each candidate track, the track finding stage has yielded a collection of hits and an estimate of the track parameters. However, the full information about the trajectory is only available at the final hit of the track (when all hits are known). To avoid biases and obtain a more precise fit, the trajectories are refitted using a Kalman smoother. Finally, the CTF outputs the tracks whose quality score (updated after the smoothing step) is above some configurable threshold. We summarize this stage in Algorithm 4.

Algorithm 4: Track Selection

```
input  : candidate tracks, cleaned by Algorithm 3
output: final reconstructed tracks
1 foreach track in candidate tracks do
2     refit the trajectory of track applying Kalman smoother;
3     calculate new quality score of track;
4     if quality score of track < threshold then
5         remove track from the set of candidate tracks;
6 output remaining candidate tracks;
```

We do not describe the Kalman smoother method in detail here, since for the purposes of our complexity analysis it is enough to know is that each candidate track is processed independently of all the others (for more information see the original reference [20]). Then, we have

Theorem 7. *Track Selection (Algorithm 4) has complexity $O(k_{clean})$, where k_{clean} is the number of tracks that passed the cleaning stage.*

3.5 Complexity of the CTF Algorithm

Combining the four stages, we find that the full CTF algorithm has complexity

$$O(n^3 + k_{\text{seed}}n + k_{\text{find}}^2 + k_{\text{clean}}), \tag{15}$$

recalling that n denotes the number of tracks in the event record and k_{seed}, k_{find}, and k_{clean} are the number of tracks that passed the seeding, finding and cleaning stages, respectively.

As we have seen in Section 3.1, it is consistent with our model that in the worst-case scenario the number of seeds k_{seed} that we generate scales like n^3. Similarly, we cannot exclude the case where $k_{\text{find}} = \Theta(n^3)$. If the allowed fraction of shared hit is large enough, the cleaning stage may not eliminate any track at all. So, k_{clean} may also scale like n^3. Taking all this into account, we conclude that

Theorem 8. *The CTF (the sequence of Algorithms 1, 2, 3, and 4) has complexity*

$$O(n^3 + k_{seed}n + k_{find}^2) = O(n^6). \tag{16}$$

4 A Classical Improvement for Track Cleaning

The track cleaning algorithm presented in Section 3.3 compares every pair of tracks coming from the finding stage. This approach does not take into account any structure of the tracks. Indeed, it would work the same if instead of calculating the fraction of shared hits we were calling a black box that outputted "clean/not clean" when given two tracks. We now present a different way to do track cleaning that takes better advantage of the structure of the problem.

We begin by describing the case where all the candidate tracks have L hits, that is, each candidate track contains exactly one hit per layer. Then, each candidate track can be uniquely identified with a vector in $\{0, \ldots, n-1\}^L$. As an example, if $L = 4$ and a given track t contains the zeroth hit from the first layer, the second hit from the second layer, the fourth hit from the third layer, and the fourth hit from the fourth layer, its corresponding *track vector* is $(0, 2, 4, 4)$. With f being the maximum allowed fraction of shared hits, define $r = \lceil fL \rceil$. We say a vector

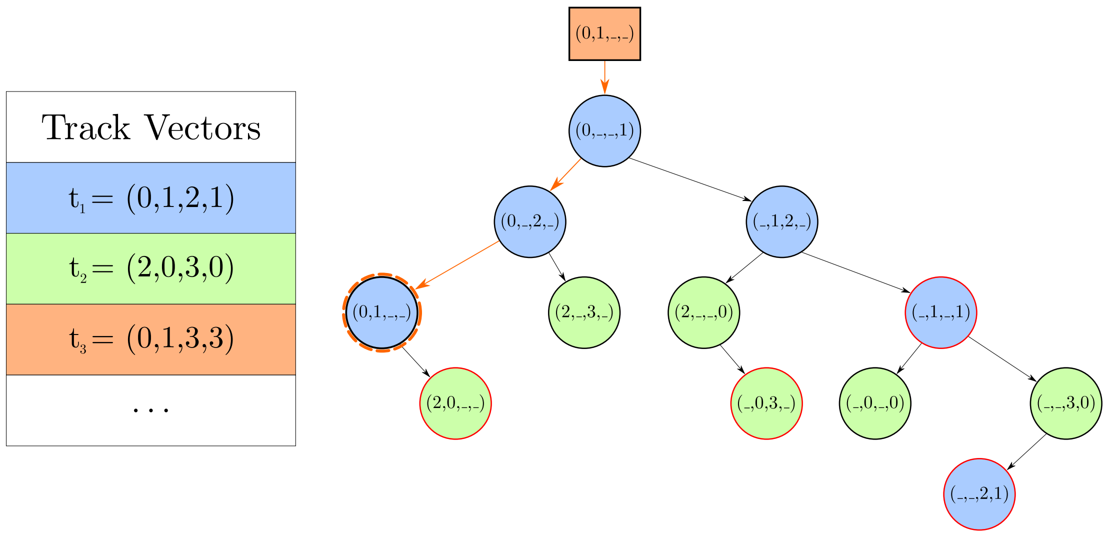


Figure 5: Track Cleaning with r-tuples tree. For this example, the first three elements of the sorted list of track vectors are $t_1 = (0, 1, 2, 1)$, $t_2 = (2, 0, 3, 0)$, and $t_3 = (0, 1, 3, 3)$. Suppose we want to exclude tracks that share two or more hits. We have build a red-black tree with the 2-tuples of t_1 and t_2 (blue and green circles, respectively). The line of the circles is red or black according to the colour of the corresponding node (see [33] for construction of red-black trees). In this illustration, we are searching for 2-tuples of t_3 in the tree. We see that 2-tuple $(0, 1, \_, \_)$ is already present in the tree – the path with orange leads to a node with that 2-tuple. So, t_3 is not going to be included in the output.

of length L is an r-tuple of a track if it is equal to the track vector at r entries and contains the symbol "_" at the others. For example, $(0, 2, \_, \_)$ and $(0, \_, 4, \_)$ are 2-tuples of the track t mentioned above. Note that there are $\binom{L}{r} = O(1)$ such r-tuples. Two tracks exceed the allowed fraction of shared hits if and only if they have (at least) r hits in common, that is, if they have a matching r-tuple.

The algorithm starts by sorting the candidate tracks by quality score. This way, if we need to discard one of two tracks we choose the one that is further down the list. We then iterate over the sorted tracks. Evidently, the first track t_1 is going to be included in the output. We create a self-balancing binary search tree $\mathcal{T}$ (like a red-black tree – see, for example, [33]) containing all of the r-tuples of t_1 (with some induced order on the r-tuples). We then move to the second track in the list t_2. For every r-tuple of t_2, we search for a match in the tree $\mathcal{T}$. If we do not find any, we insert all of t_2's r-tuples into $\mathcal{T}$ and we add t_2 to the output. Otherwise, t_2 is not included in the output and we leave the tree unchanged. We repeat this procedure for the remaining tracks. In the end, the output contains all the desired tracks. See Figure 5 for an illustration of the algorithm.

With each accepted track only $\binom{L}{r} = O(1)$ elements are inserted in $\mathcal{T}$. Since k_{find} candidate tracks reach the cleaning stage, the size of the tree never exceeds $O(k_{\text{find}})$. So, we guarantee $O(\log k_{\text{find}})$ complexity for the search and insertion tasks. This means that we only spend $O(\log k_{\text{find}})$ time per candidate track. Overall, our cleaning algorithm has complexity $O(k_{\text{find}} \log k_{\text{find}})$.

To generalize this to the case of varied number of hits per track, note that we can only find up to $L = O(1)$ different track sizes. Let $R = \lceil fL \rceil$. We initialize R^2 empty balanced binary search trees $\mathcal{T}_{i,j}$ for $i, j \in \{1, 2, \ldots, R\}$. The first track t_1 is immediately included in the output. Say it has L_1 hits and let $r_1 = \lceil fL_1 \rceil$. We insert all of the r-tuples of t_1 for $r \leq r_1$ into $\mathcal{T}_{r,r_1}$. Let the second track t_2 have size L_2 and $r_2 = \lceil fm_2 \rceil$. There are two cases to consider when two tracks share more than $\min(r_1, r_2)$ hits:

(a) $r_1 \leq r_2$: the overlapping tuples are represented in $\mathcal{T}_{r_1,r_1}$. Searching all trees $\mathcal{T}_{r,r}$ for $r \leq r_2$ will reveal the overlap.

(b) $r_1 > r_2$: the overlapping tuples are represented in $\mathcal{T}_{r_2,r_1}$. Searching all trees $\mathcal{T}_{r_2,r}$ for $r > r_2$ will reveal the overlap.

If we do not find any match, we insert all of the r-tuples of t_2 for $r \leq r_2$ into $\mathcal{T}_{r,r_2}$ and add t_2 to the output. Repeating this for all tracks will guarantee that there are no two tracks t_i and t_j in the output sharing more than $\min(r_i, r_j)$ hits. We write down our improved version of track cleaning in Algorithm 5.

Algorithm 5: Improved Track Cleaning

```
   input  : candidate tracks, generated by Algorithm 2
   output: cleaned candidate tracks
1  sort candidate tracks by quality score;
2  set $R = \lceil fL \rceil$;
3  initialize empty trees $\mathcal{T}_{i,j}$ for $i, j \in \{1, \ldots, R\}$;
4  foreach track in candidate track do
5      set $r = \lceil fL \rceil$, where $L$ is number of hits of track;
6      for $r'$ from 1 to $r$ do
7          foreach $r'$-tuple of track do
8              if $r'$-tuple is in $\mathcal{T}_{r',r'}$ then
9                  remove track from set of candidate tracks;
10     for $r'$ from $r$ to $R$ do
11         foreach $r$-tuple of track do
12             if $r$-tuple is in $\mathcal{T}_{r,r'}$ then
13                 remove track from set of candidate tracks;
14     if track has not been removed then
15         for $r'$ from 1 to $r$ do
16             foreach $r'$-tuple of track do
17                 insert $r'$-tuple into tree $\mathcal{T}_{r',r}$;
18 output remaining candidate tracks;
```

Like before, for each accepted candidate track the number of tuples inserted into the corresponding search tree s bounded by $\binom{L}{r} = O(1)$. Therefore, no tree will contain more than $O(k_{\text{find}})$ elements, and the search and insert operations can always be performed in $O(\log k_{\text{find}})$ time. Since there are $R^2 = O(1)$ trees, we spend $O(\log k_{\text{find}})$ per track candidate. We conclude that

Theorem 9. *Algorithm 5 performs the task of track cleaning (Algorithm 3) in $\tilde{O}(k_{find})$ time.*

Considering this version of track cleaning, the complexity of the CTF becomes dominated by the finding stage. We find that

Theorem 10. *The improved CTF (the sequence of Algorithms 1, 2, 5, and 4) has complexity*

$$O(n^3 + nk_{seed}) = O(n^4). \tag{17}$$

5 Quantum Algorithms for Track Reconstruction

5.1 Seed Generation with Quantum Search

We have seen in Section 3.1 that the CTF's seed generation stage takes $O(n^3)$ time (Theorem 2). We have also shown that Algorithm 1 is optimal, as in the worst-case scenario we may need to output $\Theta(n^3)$ seeds (Lemma 3). The complexity of Algorithm 1 is independent of how many seeds are actually formed. In contrast, here we show how quantum computers can reach a lower complexity if k_{seed}, the number of seeds, is $O(n^a)$ for $a < 3$.

According to the QRAM model, we assume access to a unitary $\mathbf{Q}$ acting as

$$\mathbf{Q}\left|l, j\right\rangle \left|x\right\rangle = \left|l, j\right\rangle \left|x \oplus \mathbf{m}_{l,j}\right\rangle, \tag{18}$$

where $\left|\mathbf{m}_{l,j}\right\rangle$ is a computational basis quantum state encoding the coordinates of the jth hit of layer l. We assume that the QRAM access, $\mathbf{Q}$, runs in $O(\log n)$ time. If the index j does not correspond to a hit (there may be fewer than n detections per layer), then $\left|\mathbf{m}_{l,j}\right\rangle$ holds a flag indicating so. Given a superposition $\sum_{l,j} \alpha_{l,j} \left|l, j\right\rangle \left|0\right\rangle$, applying $\mathbf{Q}$ loads the hits' data coherently

$$\mathbf{Q}\left(\sum_{l,j} \alpha_{l,j} \left|l, j\right\rangle \left|0\right\rangle\right) = \sum_{l,j} \alpha_{l,j} \left|l, j\right\rangle \left|\mathbf{m}_{l,j}\right\rangle. \tag{19}$$

We consider a unitary transformation $\mathbf{U}_{seed}$ that recognizes if a hit triplet forms a valid seed:

$$\mathbf{U}_{seed}\left|\mathbf{m}_{0,j_0}, \mathbf{m}_{1,j_1}, \mathbf{m}_{2,j_2}\right\rangle = \begin{cases} -\left|\mathbf{m}_{0,j_0}, \mathbf{m}_{1,j_1}, \mathbf{m}_{2,j_2}\right\rangle, & \text{if } (\mathbf{m}_{0,j_0}, \mathbf{m}_{1,j_1}, \mathbf{m}_{2,j_2}) \\ & \text{passes the seeding stage} \\ +\left|\mathbf{m}_{0,j_0}, \mathbf{m}_{1,j_1}, \mathbf{m}_{2,j_2}\right\rangle, & \text{otherwise.} \end{cases} \tag{20}$$

Since any classical computation can be simulated by a quantum computer [34], this is clearly possible. Moreover, because we can recognize if a hit triplet constitutes a valid seed with a $O(1)$-sized circuit, we can also build a quantum circuit for $\mathbf{U}_{seed}$ using $O(1)$ gates. Using $\mathbf{U}_{seed}$ and $\mathbf{Q}$, it is straightforward to form a unitary transformation $\mathbf{O}_{seed}$ acting on $\{\left|0\right\rangle, \left|1\right\rangle\}^{\otimes 3\log n}$ (possibly along some ancillary qubits) that marks the state $\left|j_0, j_1, j_2\right\rangle$ if the corresponding hit triplet constitutes a good seed:

$$\mathbf{O}_{seed}\left|j_0, j_1, j_2\right\rangle = \begin{cases} -\left|j_0, j_1, j_2\right\rangle, & \text{if } (\mathbf{m}_{0,j_0}, \mathbf{m}_{1,j_1}, \mathbf{m}_{2,j_2}) \text{ passes the seeding stage} \\ +\left|j_0, j_1, j_2\right\rangle, & \text{otherwise.} \end{cases} \tag{21}$$

If any of the indices $(0, j_0)$, $(1, j_1)$ or $(2, j_2)$ does not correspond to a hit, we assume that $\mathbf{O}_{seed}$ leaves the state $\left|j_0, j_1, j_2\right\rangle$ unchanged. We can run the circuit for $\mathbf{O}_{seed}$ in $O(\log(n))$ time.

We start by preparing all triplets in superposition. For simplicity, we assume that n is a power of two. Starting from the all-zero state, we can do this by applying $3\log n$ parallel single-qubit Hadamard gates (also known as the Walsh-Hadamard transform, which we denote by $\mathbf{H}$). Now define θ and m as

$$\theta = \arcsin\left(\sqrt{\frac{k_{\text{seed}}}{n^3}}\right), \quad m = \left\lfloor \frac{\pi}{4\theta} \right\rfloor, \tag{22}$$

and let $\mathbf{G}$ be the unitary transformation

$$\mathbf{G} = \mathbf{H} \cdot (2 \left|0\right\rangle \left\langle 0\right| - I) \cdot \mathbf{H} \cdot \mathbf{O}_{seed}. \tag{23}$$

From Grover's algorithm,

Theorem 11 (Grover's Search [35, 36])**.** *Let m and $\mathbf{G}$ be defined as in (22) and (23), respectively. Then, if we measure the state*

$$\mathbf{G}^m \cdot \left(\frac{1}{\sqrt{n^3}} \sum_{j_0,j_1,j_2=0}^{n-1} \left|j_0, j_1, j_2\right\rangle \right) \tag{24}$$

in the computational basis we will find a good seed (i.e., a triplet (j_0, j_1, j_2) such that $(\mathbf{m}_{0,j_0}, \mathbf{m}_{1,j_1}, \mathbf{m}_{2,j_2})$ passes the seeding stage) with probability at least $1/2$.

Preparing the state (24) involves calling the operator $\mathbf{G}$ $m = O(\sqrt{n^3/k_{\text{seed}}})$ times, representing a complexity advantage over what we could do classically. Note, however, that applying Grover's algorithm requires determining m, which we cannot do since we do not know *a priori* what is the value of k_{seed}. For that purpose, we can use the quantum counting algorithm of Brassard, Høyer, and Tapp [37].

Theorem 12 (Quantum Counting [37])**.** *There is a quantum algorithm (call it* **QCount**$(\mathbf{G})$*) that outputs k_{seed} with probability at least $3/4$, using an expected number of $O(\sqrt{n^3 k_{seed}})$ calls to* $\mathbf{G}$.

Combining these techniques, we propose performing seed generation with Algorithm 6.

Algorithm 6: Seed Generation with Quantum Search.

```
  input  : event record
  output: seeds
1 $\tilde{k} \leftarrow$ QCount(G) ;
2 set $m$ as in equation (22), with $\tilde{k}$ as an estimate for $k_{\text{seed}}$;
3 while we have not found $\tilde{k}$ good seeds do
4     prepare and measure state (24);
5     if outcome $j_0, j_1, j_2$ corresponds to a good seed then
6         add $(\mathbf{m}_{0,j_0}, \mathbf{m}_{1,j_1}, \mathbf{m}_{2,j_2})$ to output;
```

Theorem 13. *Algorithm 6 finds all good seeds with probability at least $1/2$, using an expected time $\tilde{O}(\sqrt{n^3 \cdot k_{seed}})$.*

Proof. Assume that quantum counting succeeds, that is, we have correctly estimated $\tilde{k} = k_{\text{seed}}$ in step 1 in $\tilde{O}(\sqrt{n^3 k_{\text{seed}}})$ time. The probability of sampling a *new* good seed in step 4 after having already found k of them is (Theorem 11)

$$\frac{1}{2} \frac{k_{\text{seed}} - k}{k_{\text{seed}}}. \tag{25}$$

Then, determining the expected time to find all good seeds is equivalent to the coupon collector's problem. In particular, the probability that we run step 4 more than $10 k_{\text{seed}} \log k_{\text{seed}}$ times is less than 1/4. That is, with probability at least 3/4 we spend $\tilde{O}(\sqrt{n^3 \cdot k_{\text{seed}}})$ time on loop 3-6. Since quantum counting succeeds with probability at least 3/4 (Theorem 12), Algorithm 6 outputs all seeds in $\tilde{O}(\sqrt{n^3 \cdot k_{\text{seed}}})$ time with probability no less than 1/2.

□

If $k_{\text{seed}} = O(n^a)$, then we can perform seed generation up to bounded error with complexity

$$\tilde{O}\left(n^{\frac{3+a}{2}}\right). \tag{26}$$

In the worst-case scenario, this shows no advantage over the classical algorithm (as is expected from Lemma 3). But for any $a < 3$ we reach a lower complexity than the classical seeding (Theorem 1). Considering the overall CTF algorithm (Theorem 10), we find a quantum advantage when $a < 2$. We establish that

Lemma 14. *The improved CTF algorithm with quantum seed generation (the sequence of Algorithms 6, 2, 5 and 4) has complexity $\tilde{O}\left(\sqrt{n^3 k_{seed}} + n k_{seed}\right)$. In particular, if $k_{seed} = O(n^a)$ with $a < 2$ this becomes $\tilde{O}\left(n k_{seed}\right)$, beating the $O(n^3)$ complexity of classical version (the sequence of Algorithms 1, 2, 5 and 4).*

5.2 Track Finding with a Quantum Minimum Finding

As we have seen, in the worst-case scenario the seeding and selection stages are optimal. With the improved strategy of Section 4 the cleaning stage is also optimal up to a logarithmic factor. So, the track finding stage is the one with most room for speedup in the worst-case scenario. More specifically, we have seen that $O(n)$ hits undergo the filtering step, while only at most $\lambda = O(1)$ of them form new track candidates (see Lemma 4). In this Section, we show how quantum search permits performing filtering with complexity of $\tilde{O}(\sqrt{n})$.

Suppose that we have followed a track up to layer $l-1$ according to the track finding method described in Section 3.2. In particular, we have evaluated the predicted state vector and corresponding covariance matrix. Based on this information, we can calculate predicted χ^2 value (equation (6)) for any measurement in layer l in $O(1)$ time. Let $\mathbf{O}_\chi$ be a unitary transformation that, given the index of a measurement, calculates the predicted χ^2 value of adding that measurement to the track:

$$\mathbf{O}_\chi \left|l, j\right\rangle \left|x\right\rangle = \left|l, j\right\rangle \left|x \oplus \chi^2_{l|l-1}(\mathbf{m}_{l,j})\right\rangle. \tag{27}$$

Similarly to the situation in the Section 5.1, we can build a quantum circuit for $\mathbf{O}_\chi$ using the classical circuit to compute the predicted χ^2 value and the QRAM operator $\mathbf{Q}$, requiring a total of $O(\log n)$ gates. Using $\mathbf{O}_\chi$ we can build a quantum circuit $\mathbf{O}_{find}$ that marks a state $\left|l, j\right\rangle \left|y\right\rangle$ if $\chi^2_{l|l-1}(\mathbf{m}_{l,j})$ is smaller than the threshold y

$$\mathbf{O}_{find} \left|l, j\right\rangle \left|y\right\rangle = \begin{cases} -\left|l, j\right\rangle \left|y\right\rangle, & \text{if } \chi^2_{l|l-1}(\mathbf{m}_{l,j}) < y \\ +\left|l, j\right\rangle \left|y\right\rangle, & \text{otherwise.} \end{cases} \tag{28}$$

By the quantum minimum finding algorithm of Dürr and Høyer [26], we can find the measurement $\mathbf{m}_{l,j}$ that minimizes $\chi^2_{l|l-1}$ with $O(\sqrt{n})$ calls to $\mathbf{O}_{find}$ – see Algorithm 7.

Theorem 15 (Quantum Minimum Finding [36])**.** *If there is a measurement $\mathbf{m}_{l,j}$ such that $\chi^2_{l|l-1}(\mathbf{m}_{l,j}) < \chi^2_0$, Algorithm 7 finds the measurement that minimizes $\chi^2_{l|l-1}$ with probability at least 1/2 in $\tilde{O}(\sqrt{n})$ time.*

Algorithm 7: Quantum Minimum Finding.

```
input  : prediction of track's state vector at layer $l-1$
output: $j$ such that $\mathbf{m}_{l,j}$ that minimizes $\chi^2_{l|l-1}$
1  initialize $j_0 \leftarrow$ empty;
2  set $y \leftarrow \chi^2_0$;
3  while $\mathbf{O}_{find}$ has been called less than $22.5\sqrt{n}$ times do
4      apply quantum exponential searching algorithm of [36] with initial state
       $\left(\frac{1}{\sqrt{n}}\sum_{j=0}^{n}|l,j\rangle\right)\cdot|y\rangle$ and with $\mathbf{O}_{find}$ as oracle;
5      if we find an state $|l,j\rangle$ such that $\chi^2_{l|l-1}(\mathbf{m}_{l,j}) < y$ then
6          set $j_0 \leftarrow j$;
7          set $y \leftarrow \chi^2_{l|l-1}(\mathbf{m}_{l,j})$;
8  if $j_0$ is not empty then
9      return $\mathbf{m}_{l,j_0}$
10 else
11     return "no good measurement"
```

With this result, our strategy for track finding becomes the following. Starting from a single seed, we do track finding by propagating up to λ tracks from layer to layer. For each of these tracks, we apply quantum minimum finding λ times to find the λ measurements with lowest predicted χ^2 value (after we have found a minimum of $\chi^2_{l|l-1}$ we can arbitrarily increase the χ^2 value of that measurement to ensure that we do not find it again in the following run of quantum minimum finding). Out of the up to λ^2 resulting track candidates, we select the λ ones with best quality score and continue propagating those. Note that this implies applying quantum minimum finding up to $L\lambda^2 n^3$ times, which means that the probability of correctly reproducing the result of the classical track finding decreases with n. Fortunately, we can make the probability of success bounded by always repeating the quantum minimum finding routine $O(\log n)$ times. We propose doing track finding as in Algorithm 8.

Theorem 16. *Algorithm 8 replicates the output of Track Finding (Algorithm 2) with probability at least* $1/2$ *in* $\tilde{O}(k_{seed}\cdot\sqrt{n})$ *time.*

Proof. In Algorithm 8, instead of looping over the candidate measurements at each layer (line 8 in Algorithm 2), we find the best measurements with a quantum minimum finding routine. We stop after having selected λ measurements per candidate track as we know that only up to λ tracks are kept at each layer (per seed). Each run of quantum minimum finding takes $\tilde{O}(\sqrt{n})$ time – Theorem 15. So, the result holds as long as we show that the probability of success is bounded by 1/2. The probability that we fail to select the best available measurement in steps 9-10 is upper bounded by

$$\frac{1}{3L\lambda^2 n^3}. \tag{29}$$

Then, the probability that we do not fail any of the $L\lambda^2 n^3$ times we run steps 9-10 is lower bounded by

$$\left(1-\frac{1}{3L\lambda^2 n^3}\right)^{L\lambda^2 n^3} \geq \frac{2}{3}. \tag{30}$$

□

Algorithm 8: Quantum Track Finding.

```
input  : seeds, generated by Algorithm 1; event record
output: candidate tracks
1  foreach seed do
2      initialize empty list candidate_tracks;
3      estimate initial state vector $\mathbf{p}_{2|2}$ and quality factor $q_2$ for seed;
4      add (seed, $\mathbf{p}_{2|2}$, $q_2$) to candidate_tracks;
5      foreach layer $l$ from 3 to $L-1$ do
6          foreach (track, $\mathbf{p}_{l-1|l-1}$, $q_{l-1}$) in candidate_tracks do
7              evaluate predicted measurement $\mathbf{m}_{l|l-1}$;
8              for $i$ from 1 to $\lambda$ do
9                  run quantum minimum finding (Algorithm 7) $\log\big(3L\lambda^2 n^3\big)$ times
                     (increasing the $\chi^2$ value of already used hits so not to find them again) ;
10                 from the samples of step 9, select the measurement $\mathbf{m}_{l,j}$ with lowest $\chi^2_{l|l-1}$
                     ;
11                 if $\chi^2_{l|l-1}(\mathbf{m}_{l,j}) < \chi^2_0$ then
12                     new_track ← track + $\mathbf{m}_{l,j}$;
13                     form new candidate track for seed with $\mathbf{m}_{l,j}$;
14                     evaluate $\mathbf{p}_{l|l}$ and quality factor $q_l$ for new_track;
15                     add (new_track, $\mathbf{p}_{l|l}$, $q_l$) to candidate_tracks;
16             if no new candidate track was formed then
17                 new_track ← track + $\mathbf{m}_{l|l-1}$;                     /* ghost hit */
18                 evaluate $\mathbf{p}_{l|l}$ and quality factor $q_l$ for new_track;
19                 add (new_track, $\mathbf{p}_{l|l}$, $q_l$) to candidate_tracks;
20             remove (track, $\mathbf{p}_{l-1|l-1}$, $q_{l-1}$) from candidate_tracks;
21         select the best $\lambda$ tracks of candidate_tracks ;             /* sorting by $q_l$ */
22     add elements of candidate_tracks to output;
```

In the worst-case scenario k_{seed} scales as n^3 (Lemma 3), so Algorithm 8 has complexity $\tilde{O}\left(n^{3.5}\right)$. Comparing with the classical case (Theorem 10), we see in this scenario a $\sqrt{n}$ quantum advantage is the best we can achieve (again, omitting poly-logarithmic factors). If we combine Algorithm 8 with the quantum seed generation algorithm developed in Section 5.1 (Algorithm 6), we get a more general result that also depends on the number of formed seeds, k_{seed}. Namely, we establish:

Theorem 17. *The improved CTF algorithm with quantum track finding (the sequence of Algorithms 1, 8, 5 and 4) has complexity*

$$\tilde{O}\left(\sqrt{n^3 k_{seed}} + \sqrt{n} k_{seed}\right) = \tilde{O}\left(n^{3.5}\right). \tag{31}$$

This beats the classical version (the sequence of Algorithms 1, 2, 5 and 4).

5.3 Reconstructing Tracks in Superposition

We saw in Section 3.5 that the size of the output (k_{clean}) may scale like n^3. For a fixed quality threshold in the selection stage, we have no guarantee that we will filter most of the candidates. In this section, we introduce a promise that the CTF only outputs $O(n)$ tracks. Intuitively, this promise means that the CTF can be applied in the asymptotic regime while keeping a constant fraction of fake tracks, i.e., tracks that do not correspond to a real charged particle. Although we do not have a natural way to guarantee this within our model (that is why we formulate it as a promise), we point out that, in practice, particle physicists empirically adjust the parameters of the tracking software according to the luminosity regime to obtain a reasonable fake track rate for most events. We will show how we can exploit this promise to construct a quantum algorithm with $\tilde{O}(n^3)$ complexity.

The promise that only $O(n)$ tracks are to be found among $O(n^3)$ track candidates suggests the use of quantum search, as in Section 5.1. This is complicated for two reasons: (a) track finding (Section 3.2) in CTF *forgets* information by selecting only some track candidates in each layer, i.e., it is not reversible, while quantum search relies on unitary transformations that are reversible; (b) track cleaning (Sections 3.3 and 4) for each track candidate depends on information about other tracks. To rectify point (a) we apply the principle of deferred measurements [34] to create a sequence of unitary transformations that mimic the CTF algorithm. To rectify point (b) we adapt the improved track cleaning from Section 4.

To construct the unitary transformations used in quantum search, we first slightly adjust the steps in the classical CTF track finding algorithm (Algorithm 2). For a given seed, CTF selects up to λ track candidates in each layer to propagate to the next layer. If fewer track candidates have acceptable χ^2 value, fewer than λ track candidates are formed. Here we form exactly λ new track candidates for every given track candidate. If there is at least one hit with $\chi_l^2(\mathbf{m}_{l,j}) < \chi_0^2$, we use λ hits with the lowest χ^2 values to build the new track candidates. If there is no such hit, we use one ghost hit and $\lambda - 1$ hits with the lowest χ^2 values. We also build tracks for all triplets in the seeding layer. This would add substantial unnecessary work in the classical case, but does not add complexity if performed in quantum superposition. As in other sections, we can add placeholder hits to ensure that each layer has exactly n hits and all tracks traverse through all L layers. Thus at the end of the track finding phase we have exactly $\lambda^L n^3$ track candidates.

Based on the modified algorithm we construct a family of unitary transformations $\mathbf{U}_i$ that perform seeding, track finding, cleaning and selection in superposition with the following effect:

$$\mathbf{U}_i \left|0\right\rangle = \sqrt{\frac{1-\epsilon}{\lambda^L n^3}} \left(\sum_{j=0}^{k_i - 1} \left|\psi_j\right\rangle \left|-q_{L-1,j}\right\rangle + \sum_{j=k_i}^{\lambda^L n^3 - 1} \left|\psi_j\right\rangle \left|+\infty\right\rangle \right) + \sqrt{\epsilon} \left|\psi_\epsilon\right\rangle \left|+\infty\right\rangle. \tag{32}$$

Here $|\psi_0\rangle, |\psi_1\rangle, \ldots, |\psi_{k_i-1}\rangle$ are the computational basis states encoding track candidates that (a) the classical CTF algorithm would output after the track finding stage and (b) do not share too many hits with the i tracks already added to the output (Section 4). $q_{L-1,j}$ is the quality score (12) at the last layer of the track encoded in ψ_j. We consider $-q_{L-1,j}$ to formulate the task as a minimization problem. The computational basis states $|\psi_{k_i}\rangle, |\psi_{k_i+1}\rangle, \ldots, |\psi_{\lambda^L n^3-1}\rangle$ encode some track candidates that do not pass (a) or (b). Note that the tracks that were previously sampled belong to this set of states. "$+\infty$" is a large positive value, so the minimum finding gives answers only in the useful subset of the entangled computational basis states. $|\psi_\epsilon\rangle$ is some arbitrary quantum state, and ϵ is the *error* probability of $\mathbf{U}_i$, i.e., the probability that the measurement of $\mathbf{U}_i|0\rangle$ would produce a result other than one of $\psi_0, \psi_1, \ldots, \psi_{\lambda^L n^3-1}$.

Next we show that such a family of unitary transformations can indeed be constructed. We first consider track finding (Lemma 18) and track cleaning (Lemma 19) sub-procedures. Track finding prepares an equal superposition over the $\lambda^L n^3$ track candidates with an additional arbitrary quantum state (33) representing the error probability of the algorithm. Selection of the k_{find} track candidates of the original CTF after track finding step is subsumed by the track selection stage (Theorem 20).

Lemma 18 (Track finding in superposition)**.** *There exists a unitary transformation* $\mathbf{U}_{\text{find}}$ (33) *that performs track finding in* $\tilde{O}(\sqrt{n})$ *time in superposition.*

$$\mathbf{U}_{\text{find}}|0\rangle = \sqrt{\frac{1-\epsilon}{\lambda^L n^3}} \sum_{j=0}^{\lambda^L n^3-1} |\psi_j\rangle + \sqrt{\epsilon}\,|\psi_\epsilon\rangle \tag{33}$$

Proof. Preparing an equal superposition over all the possible seed triplets, i.e., seeding (implicit in $\mathbf{U}_{\text{find}}$), can be done in $\tilde{O}(1)$ time with the Walsh-Hadamard transform. We saw in Section 5.2 that we can perform track finding for one seed in $\tilde{O}(\sqrt{n})$ time with constant probability. However, both quantum minimum finding and its sub-procedure – quantum exponential searching algorithm – use measurements. While we cannot use measurements in our unitary transformations $\mathbf{U}_{\text{find}}$, we can apply the principle of deferred measurements [34]. Whenever the algorithm in Section 5.2 performs a measurement, we can instead perform CNOT operations on an ancillary register. When the algorithm conditions a quantum operation on a measurement result, we can perform a controlled operation with the ancillary register as the control. The probability $(1-\epsilon)$ to get the desired result based on measurements during the procedure or by deferring the measurement is the same. We can replicate the randomness in the quantum exponential searching algorithm [36] by conditioning operations on the equal superposition of the allowed values $\{0, 1, \ldots, m\}$, where m is an arbitrary integer. By conditioning on the digits of the binary representation of these values, as in quantum counting [37], we can ensure that we only need $O(m)$ such operations and the asymptotic computational complexity of quantum exponential searching algorithm remains unchanged (up to constant factors).

One issue with this approach is that both the number of iterations of the outer loop of the quantum minimum finding (Algorithm 7) and the time complexity of its sub-procedure – quantum exponential searching [36] – may be proportional to $\sqrt{n}$. In the classical algorithm, if the quantum exponential searching takes more time, we can limit the number of iterations of the outer loop to ensure running time $\tilde{O}(\sqrt{n})$. In the quantum circuit we need to account for the worst case number of iterations in the main loop and the worst-case running time in the sub-procedure. This requires more than $\tilde{O}(\sqrt{n})$ gates. However, we can set a hard limit on the number of iterations of the main loop. Since the expected size of the search space decreases by more than a half with each iteration of the outer loop, the expected number of iterations to reach the minimum is less than $\log n + 1$. If we limit the number of iterations of the outer loop to $c(\log n + 1)$ for some constant c, then by Markov's inequality the probability that we have not reached the minimum is less than $1/c$. Since it is still upper-bounded by a constant, the rest of the analysis does not change. The limit on the number of iterations also implies a limit on the number of measurements and the required number of ancillary registers to account for the measurements in the quantum procedure.

□

Lemma 19 (Track cleaning in superposition)**.** *There exists a unitary transformation that runs in $\tilde{O}(1)$ time and marks the track candidates that do not share any r-tuple of hits with any track already added to the output.*

Proof. We have assumed that all particles traverse all layers, so all tracks are of length L and are allowed to share up to exactly r hits for some value of r. As in Section 4, we can generalize it to variable length tracks with a constant factor increase in complexity. Like in the classical case, we can test whether an r-tuple has already been added to the tree $\mathcal{T}$ with $O(\log n)$ queries to QRAM storing the values of the nodes of tree $\mathcal{T}$. Thus each track can be associated with a list of $\binom{L}{r}$ binary values indicating if an r-tuple has already been added to the output in $\tilde{O}(1)$ time. Testing whether any of these values is equal to 1 requires $O(1)$ gates. Hence the total time required to mark the necessary track candidates is $\tilde{O}(1)$. □

Lemma 20. *Each unitary transformation $\mathbf{U}_i$ (32) can be built to run in time $\tilde{O}(\sqrt{n})$.*

Proof. We already saw that track finding requires $\tilde{O}(\sqrt{n})$ time (Lemma 18) and testing, whether a track overlaps with any already added to the output takes $\tilde{O}(1)$ time (Lemma 19). Procedures necessary for track selection – marking the tracks that pass the track finding stage in CTF, refitting, recalculating the score and comparing to a threshold value – depend on a constant number of fixed-precision numbers, and hence can be done in $\tilde{O}(1)$ time. So the time complexity of $\mathbf{U}_i$ is dominated by the track finding and is $\tilde{O}(\sqrt{n})$. □

We will now describe how we can use transformations $\mathbf{U}_i$ with quantum minimum finding to reconstruct the tracks one-by-one (Algorithm 9). We will search for $\psi_i^* = \arg\min Q_i(\psi)$, where $Q_i(\psi)$ is the score encoded in the second register of $\mathbf{U}_i |0\rangle$ (32). The time complexity of the quantum minimum finding [26] remains the same (up to constant factors) if instead of quantum exponential searching [36] we use amplitude amplification (Theorem 21).

Theorem 21 (Amplitude amplification [38])**.** *Let $\mathcal{A}$ be any quantum algorithm that uses no measurements, and let a denote the initial success probability of $\mathcal{A}$. There exists a quantum algorithm that finds a good solution using an expected number of applications of $\mathcal{A}$ and $\mathcal{A}^{-1}$ which are in $\Theta(1/\sqrt{a})$ if $a > 0$, and otherwise runs forever.*

Let a be the probability to find ψ_i^* (or any specific track encoded in $\{\psi_0, \psi_1, \ldots, \psi_{k_i-1}\}$) by measuring $\mathbf{U}_i |0\rangle$. Then $a = (1-\epsilon)/(\lambda^L n^3)$ and the expected number of calls to $\mathbf{U}_i$ by the quantum minimum finding algorithm for a constant probability of error is

$$O\left(\sqrt{(\lambda^L n^3)/(1-\epsilon)}\right) = O\left(\sqrt{n^3}\right). \tag{34}$$

As in Section 5.2, repeating the quantum minimum finding algorithm $O(\log n)$ times allows us to reduce the error probability to $O(1/n)$ so that sampling $O(n)$ tracks has a constant probability of error. In particular, since one application of the quantum minimum finding algorithm ensures failure probability smaller than 1/2 and there cannot be more than λn^3 tracks, the probability to not find the minimum in any of the iterations (if there are any valid tracks remaining) after repeating the algorithm $\lceil \log 2\lambda n^3 \rceil$ times is below 1/2.

Once we have found the best track, we can build a self-balancing binary tree $\mathcal{T}$ (as in Section 4) to be used in track selection for the next track. More generally – suppose that we have found the j best tracks tracks that the CTF algorithm outputs. Each time we find a new track, we insert it in $\mathcal{T}$. This tree never exceeds $O(n)$ size, and so the insertion operation cost is $O(\log n)$. $\mathbf{U}_i$ queries $\mathcal{T}$ to mark as invalid those tracks that have an overlap with the i tracks already added to output.

Algorithm 9: Track reconstruction in superposition.

```
   input  : event record
   output: reconstructed tracks
1  initialize empty self-balancing binary tree $\mathcal{T}$;
2  initialize $i \leftarrow 0$;
3  repeat
4      for $j$ from 1 to $\lceil \log(2\lambda n^3) \rceil$ do
5          track$_j$ ← output of the quantum minimum finding algorithm minimizing $Q_i(\psi)$ –
            the score encoded in the second register of $\mathbf{U}_i |0\rangle$ (32);
6      track ← $\arg\min_j Q_i($track$_j)$;
7      if track passes CTF criteria and none of its $r$-tuples are in $\mathcal{T}$ then
8          foreach $r$-tuple of track do
9              insert $r$-tuple into tree $\mathcal{T}$;
10         add track to output;
11         $i \leftarrow i + 1$;
12     else
13         stop
```

Theorem 22. *Algorithm 9 reconstructs $O(n)$ tracks in time $\tilde{O}(n^3)$.*

Proof. Each iteration of the main loop in Algorithm 9 takes $\tilde{O}(\sqrt{n^3} \max_i T_{\mathbf{U}_i})$ time, where $\max_i T_{\mathbf{U}_i} = \tilde{O}(\sqrt{n})$ (Lemma 20). There are $O(n)$ iterations to reconstruct $O(n)$ tracks. Thus the total time complexity of Algorithm 9 is

$$\tilde{O}\left(n\sqrt{n^3}\sqrt{n}\right) = \tilde{O}\left(n^3\right). \tag{35}$$

□

For the special case where tracks are not allowed to share any hits, the approach described in this section allows the complete removal of the cleaning stage. Once the best track is found, all the points that belong to it can be masked (removed) and the algorithm is run again to find the best track on the remaining points.

6 Discussion About our Assumptions

In our analysis we have simplified some aspects of the CTF algorithm in order to make the description clearer. We now comment on some of the omitted details.

First, we should note that in the CTF the tracks are reconstructed by multiple iterations. That is, the sequence of four stages described in Section 3 (seeding, finding, cleaning, and selection) is called several times for the same event record. The idea of this iterative tracking is that the initial iterations search for the tracks that are easiest to find (high transverse momentum, and produced near the interaction region). After each iteration, the hits associated to the reconstructed tracks are removed, thereby simplifying the subsequent iterations. As far as our complexity analysis is concerned, the most significant modification from iteration to iteration is the number of hits used to form seeds. The first iteration forms seeds with hit triplets, as we have described in Section 3.1. But in some subsequent iterations seeds are formed by picking just two hits, as we can use the results of the previous iteration to reconstruct the collision vertices, which serve as the "third hit". For these iterations, assuming that there is a constant number of collision vertices, the complexity of seed generation becomes $O(n^2)$.

Regarding our model for the generation of the trajectories, we assumed that the probability distribution p_π on parameter space $\mathcal{P}$ was strictly positive. This was not deduced from explicit particle physics calculations, but is a rather lax assumption that includes the seemingly reasonable assumption that no scattering direction is forbidden. We've also assumed that an event is generated by drawing n random samples $\pi_1, \ldots \pi_n$, each following p_π. Underlying this there is the physical assumption that the trajectories of the charged particles are treated as uncorrelated.

Another of our assumptions was that the layers were continuous surfaces, as this made the description of the algorithm clearer (especially for the track finding stage – Section 3.2). In reality, the layers are formed by overlapping sensor modules. This means that it is possible for a particle to leave more than one detection per layer. To accommodate this possibility, at each layer the CTF selects compatible modules, which are the ones whose boundaries are up to a given distance from the predicted measurement. These modules are divided into module groups in such a way that no two modules in the same group overlap. Only the best measurement from each group is considered to integrate the track candidate. Again, we only allow up to five new track candidates per step. At each iteration there are $O(1)$ module groups, each with $O(n)$ hits, so the claimed complexity remains the same.

Furthermore, we assumed that the detector was a collection of $L = O(1)$ cylindrical layers. In reality, the CMS tracker has a barrel-like shape, with thirteen cylindrical layers aligned with the beam line and fourteen disk layers in the transverse plane. We did not include the detailed geometry of the detector in our discussion in order to simplify the exposition. In fact, our analysis holds for any disposition of layers as long as we assume that each particle only traverses one layer at a time and that their trajectories do not return to a previously visited layer.

We have considered that the hit data is given in the form of three-dimensional points. Actually, as a charged particle traverses a layer, it activates multiple sensor pixels. Then, the signals in neighbouring pixels are grouped together to form three-dimensional clusters. The centroid of each cluster determines a hit's position. But the cluster shape also carries information. In particular, in some cases it is possible to exclude a hit from a given track based on the incompatibility between the hit's cluster shape and the track's trajectory. We may see this as a motivation to think about the case where $k_{\text{seed}} = O(n^a)$ with $a < 3$, as in Section 5.1 – even though the cluster shape information does not provide a mean to find the good seeds faster, it guarantees that we can recognize them.

In summary, we see that several of our simplifying assumptions could be lifted without changing our conclusions. Arguably, the strongest assumption was ignoring the hits' cluster shape information, as that might be used to exclude $k_{\text{seed}} = O(n^3)$ as a worst-case scenario.

As a final note, we would like to point out that more recent versions of the CTF have been developed [21]. The most important modifications concern the seed generation stage, which now uses quadruplets of hits instead of triplets. The complexity of the other stages should remain as we have described.

Table 1: Summary of the computational complexity of the Combinatorial Track Finder (CTF) algorithm. n is the number of charged particles present in the event record, k_{seed} is the number of seeds generated, k_{find} is the number tracks built during the track finding stage, and k_{clean} is the number of tracks that pass the cleaning stage. Note that $k_{\text{seed}} = O(n^3)$, $k_{\text{find}} = O(k_{\text{seed}})$, $k_{\text{clean}} = O(k_{\text{find}})$. In the *General Case* column we show the complexity of the algorithms for each of the track reconstruction stages, both the classical and quantum versions. The two rows for the track cleaning stage refer the original version of [20] and to the one that we propose on Section 4. On the quantum side some entries are marked as "–" where we did not propose/expect a quantum algorithm with advantage over the classical one. In the final row we write the complexity of the full CTF algorithm, both in the worst-case (where $k_{\text{seed}} = O(n^3)$) and in the particular case where the number of seeds is $O(n)$. We find a quantum speedup for any value of k_{seed}. In the final column we write the complexity of our algorithm for track reconstruction under the promise that the total number of reconstructed tracks is $O(n)$.

Stages of CTF		**General Case**		**Promise of $O(n)$ Reconstructed Tracks**
		Classical	**Quantum**	**Quantum**
Seed Generation		$O(n^3)$ (Theorem 2)	$\tilde{O}(\sqrt{n^3 k_{seed}})$ (Theorem 13)	$\tilde{O}(n^3)$ (Theorem 22)
Track Finding		$O(nk_{seed})$ (Theorem 5)	$\tilde{O}(\sqrt{n}k_{seed})$ (Theorem 16)	
Track Cleaning (original)		$O(k_{find}^2)$ (Theorem 6)	–	
Track Cleaning (improved)		$\tilde{O}(k_{find})$ (Theorem 9)	–	
Track Selection		$O(k_{clean})$ (Theorem 7)	–	
Full Algorithm (improved)	Worst-Case ($k_{\text{seed}} = O(n^3)$)	$O(n^4)$ (Theorem 10)	$\tilde{O}(n^{3.5})$ (Theorem 14)	
	$O(n)$ Seeds	$O(n^3)$ (Theorem 10)	$\tilde{O}(n^2)$ (Theorem 14)	

7 Conclusions

We have analyzed the computational complexity of the state-of-the-art classical tracking algorithm *Combinatorial Track Finder* (CTF) as a function of the number of charged particles n produced in a given event. In particular, we have analyzed separately the different stages of CTF: seed generation, track finding, track cleaning, and track selection. Furthermore, we have developed quantum algorithms offering a speedup for the seed generation and track finding stages. Our results are summarized in Table 1, and we discuss here our main findings.

On the classical front, we have shown that in the worst-case scenario the CTF algorithm has complexity $O(n^6)$, as stated in Theorem 8. We have then proposed a modification to the track cleaning stage, namely Algorithm 5, that lowers the corresponding complexity to:

$$O\left(n^3 + nk_{\text{seed}}\right), \tag{36}$$

where k_{seed} is the number of formed track seeds, as stated in Theorem 10. In the worst-case scenario, where $k_{\text{seed}} = O(n^3)$ as indicated in Lemma 3, our modified CTF algorithm scales globally as $O(n^4)$. We have thus found a classical improvement to the current CTF algorithm.

Then, on the quantum front, we have shown that, by combining our proposed quantum search routines (Algorithms 6 and 8), we can reconstruct the same tracks as the classical CTF algorithm up to bounded error probability in time:

$$\tilde{O}\left(\sqrt{n^3 k_{\text{seed}}} + \sqrt{n} k_{\text{seed}}\right), \tag{37}$$

as stated in Theorem 17. This represents an advantage over the classical complexity for any value of k_{seed}. In particular, in the worst-case scenario, our quantum version of CTF scales globally as $\tilde{O}(n^{3.5})$, offering a speedup, albeit small, with respect to our improved classical version.

Finally, we have shown that under the reasonable assumption that the fraction of fake tracks is kept constant in the asymptotic limit – that is, the number of reconstructed tracks is $O(n)$ – we can replicate the output of the CTF algorithm (Algorithm 9) with quantum complexity:

$$\tilde{O}(n^3), \tag{38}$$

as stated in Theorem 22.

Overall, our comprehensive computational complexity analysis led us to a classical improvement of the Combinatorial Track Finder algorithm, as well as, to the best of our knowledge, to the first proof of a quantum speedup for a state-of-the-art track reconstruction method. And, even though asymptotic results may be of limited use for practical problems, and quantum hardware may still be far from being able to address big data problems, we hope our original approach to tracking can motivate further investigations on the potential of quantum computation to tackle the increasingly challenging, and potentially intractable classically, High-Energy Physics data analysis problems.

Acknowledgements

The authors thank Felice Pantaleo for precious discussions about the classical Combinatorial Track Finder algorithm. Furthermore, the authors acknowledge project *QuantHEP – Quantum Computing Solutions for High-Energy Physics*, supported by the EU H2020 QuantERA ERA-NET Cofund in Quantum Technologies, and FCT – Fundação para a Ciência e a Tecnologia (QuantERA/0001/2019). DM, AK, SP, GQ, PB, JS, YO thank the support from FCT, namely through project UIDB/50008/2020. DM acknowledges the support from FCT through scholarship 2020.04677.BD. AG has been partially supported by National Science Center under grant agreement 2019/32/T/ST6/00158 and 2019/33/B/ST6/02011. MK thanks MikroTik for the scholarship administrated by the UL Foundation. SP thanks the support from the la Caixa foundation through sholarship LCF/BQ/DR20/11790030. GQ thanks the support from FCT through project CEECIND/02474/2018.

References

[1] Johannes Albrecht et al. “A Roadmap for HEP Software and Computing R&D for the 2020s”. In: *Computing and Software for Big Science* 3.1 (Mar. 2019), p. 7. DOI: `10.1007/s41781-018-0018-8`.

[2] “High-Luminosity Large Hadron Collider (HL-LHC): Technical Design Report V. 0.1”. In: 4 (Nov. 2017). Ed. by Giorgio Apollinari et al. DOI: `10.23731/CYRM-2017-004`.

[3] Asmaa Abada et al. “FCC Physics Opportunities: Future Circular Collider Conceptual Design Report Volume 1”. In: *Eur. Phys. J. C* 79.6 (June 2019), p. 474. DOI: `10.1140/epjc/s10052-019-6904-3`.

[4] Frédéric Bapst et al. “A Pattern Recognition Algorithm for Quantum Annealers”. In: *Computing and Software for Big Science* 4.1 (Dec. 2020), p. 1. ISSN: 2510-2036. DOI: `10.1007/s41781-019-0032-5`.

[5] Alexander Zlokapa et al. *Charged Particle Tracking with Quantum Annealing-Inspired Optimization.* Aug. 2019. arXiv: `1908.04475 [quant-ph]`.

[6] Souvik Das et al. *Track clustering with a quantum annealer for primary vertex reconstruction at hadron colliders.* Mar. 2019. arXiv: `1903.08879 [hep-ex]`.

[7] Cenk Tüysüz et al. *Particle Track Reconstruction with Quantum Algorithms.* Mar. 2020. arXiv: `2003.08126 [quant-ph]`.

[8] Cenk Tüysüz et al. *A Quantum Graph Neural Network Approach to Particle Track Reconstruction.* July 2020. arXiv: `2007.06868 [quant-ph]`.

[9] Cenk Tüysüz et al. *Performance of Particle Tracking Using a Quantum Graph Neural Network.* Dec. 2020. arXiv: `2012.01379 [quant-ph]`.

[10] Wen Guan et al. “Quantum machine learning in high energy physics”. In: *Machine Learning: Science and Technology* 2.1 (Mar. 2021), p. 011003. DOI: `10.1088/2632-2153/abc17d`.

[11] Sau Lan Wu et al. *Application of Quantum Machine Learning using the Quantum Variational Classifier Method to High Energy Physics Analysis at the LHC on IBM Quantum Computer Simulator and Hardware with 10 qubits.* Dec. 2020. arXiv: `2012.11560`.

[12] Alex Mott et al. “Solving a Higgs optimization problem with quantum annealing for machine learning”. In: *Nature* 550.7676 (Oct. 2017). DOI: `10.1038/nature24047`.

[13] Koji Terashi et al. *Event Classification with Quantum Machine Learning in High-Energy Physics.* Feb. 2020. arXiv: `2002.09935 [physics.comp-ph]`.

[14] Sau Lan Wu et al. *Application of Quantum Machine Learning using the Quantum Variational Classifier Method to High Energy Physics Analysis at the LHC on IBM Quantum Computer Simulator and Hardware with 10 qubits.* Dec. 2020. arXiv: `2012.11560 [quanth-ph]`.

[15] Jamie Heredge et al. *Quantum Support Vector Machines for Continuum Suppression in B Meson Decays.* Mar. 2021. arXiv: `2103.12257 [quanth-ph]`.

[16] Vasileios Belis et al. *Higgs analysis with quantum classifiers.* Apr. 2021. arXiv: `2104.07692 [quanth-ph]`.

[17] Anthony E. Armenakas and Oliver K. Baker. *Application of a Quantum Search Algorithm to High- Energy Physics Data at the Large Hadron Collider.* Oct. 2020. arXiv: `2010.00649 [quant-ph]`.

[18] Su Yeon Chang et al. *Quantum Generative Adversarial Networks in a Continuous-Variable Architecture to Simulate High Energy Physics Detectors.* Jan. 2021. arXiv: `2101.11132 [quanth-ph]`.

[19] Annie Y. Wei et al. "Quantum algorithms for jet clustering". In: *Phys. Rev. D* 101.9 (May 2020), p. 094015. DOI: `10.1103/PhysRevD.101.094015`.

[20] The CMS Collaboration. "Description and performance of track and primary-vertex reconstruction with the CMS tracker". In: *Journal of Instrumentation* 9.10 (Oct. 2014), p. 10009. DOI: `10.1088/1748-0221/9/10/P10009`.

[21] Andrea Bocci et al. "Heterogeneous Reconstruction of Tracks and Primary Vertices With the CMS Pixel Tracker". In: *Frontiers in Big Data* 3 (Dec. 2020), p. 49. DOI: `10.3389/fdata.2020.601728`.

[22] Rudolf Frühwirth. "Application of Kalman filtering to track and vertex fitting". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 262.2 (Dec. 1987), pp. 444–450. DOI: `https://doi.org/10.1016/0168-9002(87)90887-4`.

[23] Giacomo Sguazzoni. "Track reconstruction in CMS high luminosity environment". In: *Nuclear and Particle Physics Proceedings* 273-275 (May 2016), p. 2497. DOI: `10.1016/j.nuclphysbps.2015.09.437`.

[24] The ATLAS Collaboration. "Performance of the ATLAS track reconstruction algorithms in dense environments in LHC Run 2". In: *European Physical Journal C* 77.10 (Oct. 2017). DOI: `10.1140/epjc/s10052-017-5225-7`. eprint: `arXiv:1704.07983v2`.

[25] Nils Braun. *Combinatorial Kalman Filter and High Level Trigger Reconstruction for the Belle II Experiment.* Springer Theses. Cham: Springer International Publishing, July 2019. DOI: `10.1007/978-3-030-24997-7`.

[26] Christoph Durr and Peter Hoyer. *A Quantum Algorithm for Finding the Minimum.* July 1996. arXiv: `quant-ph/9607014 [quant-ph]`.

[27] Ryan Babbush et al. "Focus beyond Quadratic Speedups for Error-Corrected Quantum Advantage". In: *PRX Quantum* 2 (1 Mar. 2021). DOI: `10.1103/PRXQuantum.2.010103`.

[28] Rainer Mankel. "Pattern recognition and event reconstruction in particle physics experiments". In: *Reports on Progress in Physics* 67.4 (Mar. 2004), p. 553. DOI: `10.1088/0034-4885/67/4/r03`.

[29] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. "Quantum Random Access Memory". In: *Physical Review Letters* 100.16 (Apr. 2008), p. 160501. DOI: `10.1103/PhysRevLett.100.160501`.

[30] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. "Architectures for a quantum random access memory". In: *Physical Review A* 78.5 (Nov. 2008), p. 052310. DOI: `10.1103/PhysRevA.78.052310`.

[31] T. Miao et al. *Beam position determination using tracks.* Aug. 2007. CERN-CMS-NOTE: 2007-021,FERMILAB-FN-0816-E.

[32] Rudolph Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *Transactions of the ASME–Journal of Basic Engineering* 82.Series D (Mar. 1960), p. 36. DOI: 10.1115/1.3662552.

[33] Thomas H. Cormen et al. *Introduction to Algorithms.* MIT Press, July 2010. ISBN: 97802620 33848.

[34] Michael Nielsen and Isaac Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, Mar. 2010. DOI: 10.1017/CBO9780511976667.

[35] Lov Grover. "Quantum Mechanics Helps in Searching for a Needle in a Haystack". In: *Phys. Rev. Lett.* 79.2 (July 1997), p. 325. DOI: 10.1103/PhysRevLett.79.325.

[36] Michel Boyer et al. "Tight Bounds on Quantum Searching". In: *Fortschritte der Physik* 46.4-5 (June 1998), p. 493. DOI: 10.1002/(sici)1521-3978(199806)46:4/5<493::aid-prop493>3.0.co;2-p.

[37] Gilles Brassard, Peter Høyer, and Alain Tapp. "Quantum counting". In: *Lecture Notes in Computer Science* 1443 (July 1998), p. 820. DOI: 10.1007/bfb0055105.

[38] Gilles Brassard et al. "Quantum amplitude amplification and estimation". In: *Quantum Computation and Information* 305 (2002), p. 53. DOI: 10.1090/conm/305.