

Large-Scale Multiple Target Tracking with Sparse Observations: Data Structures and Algorithms for Asteroid Tracking

Thesis Proposal

Jeremy Kubica

December 17, 2004

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

© Carnegie Mellon University

Abstract

Detecting and tracking asteroids from observational data is an important, but extremely computationally expensive, task. The goal is to link together observations from different time steps into sets of observations that correspond to the same asteroid. These linkages can then be used to determine the orbits for new asteroids, attribute the observations to previously known asteroids, and assess the potential risk posed by the object.

What sets this domain apart from previous large-scale multiple target tracking problems is the sheer scale of the problem and the sparse nature of the observation schedule. New sky surveys allow us to observe increasing faint objects, providing millions of true and noise observations in the process. Further, detections of a given object may be sparse, sporadic, and widely spaced. Each individual viewing will only cover a small fraction of the sky and there may be a significant gap in time before that area is observed again.

This work deals with the computational issues inherent to large-scale target tracking and the development of techniques and algorithms that mitigate or eliminate these issues. I examine two fundamental issues to tracking in the asteroid domain: tracking with a large number of both true and noise observations and tracking under a sparse observation schedule. I propose new data structures, algorithms, and general approaches to efficiently and accurately deal with these issues.

Contents

1	Introduction	1
2	Problem Overview	2
2.1	Computational Challenges and Contributions	3
2.1.1	Initiation	3
2.1.2	Prediction	4
2.1.3	Data Association	4
2.1.4	Evidence Incorporation and Maintenance	4
2.1.5	Trajectory-Based Spatial Queries	4
2.2	Notation	5
3	The Asteroid Domain	5
3.1	Brief Overview of Asteroid Orbits	6
3.2	Asteroid Linkage Requirements	7
3.3	Previous Approaches	8
3.4	The Need for Better Linkage Algorithms	8
4	Current and Completed Work: Initial Asteroid Linkage Algorithm	9
4.1	Simplified Multiple Hypothesis Tracking	9
4.2	The Use of Quadratic Tracks	11
4.3	The Use of Spatial Structure	11
4.4	Results	11
4.5	Discussion and Future Directions	13
5	Current and Completed Work: Efficient Track Initiation	13
5.1	Problem Definition	14
5.2	Previous Approaches	15
5.2.1	Sequential Tracking and Track Initiation	16
5.2.2	Parameter Space Methods	16
5.3	Multiple Tree Algorithm	17
5.4	Sample Results	18
5.4.1	Algorithms	18
5.4.2	Astronomy Data	18
5.4.3	Results	20
5.4.4	Discussion	21
6	Current and Completed Work: Queries on Trajectories	21
6.1	Query Types	21
6.1.1	Occurrence Queries	21
6.1.2	Attribution Queries	22
6.1.3	Precovery Query	22
6.2	Previous Approaches	22
6.3	Adaptations and New Algorithms	23
6.3.1	Brute Force Computation	23

6.3.2	KD-tree of Plates	23
6.3.3	KD-tree of Tracks	24
6.3.4	Ball-tree of Tracks	25
6.3.5	Combining Trees	26
6.4	Sample Results	26
7	Research Plan and Future Work	28
7.1	Contributions	28
7.2	Timeline	29

1 Introduction

Asteroid detection and tracking provides a tracking problem of unprecedented scale. Ideally we would like to identify and track all asteroids that are large enough to penetrate the earth's atmosphere and cause significant damage upon impact. This would then allow us to accurately determine their orbits so that we can assess their risk. New astronomical surveys, designed specifically for this purpose, will provide a wealth of observational data to this end. Further, the use of new equipment and digital processing provide the opportunity to track increasingly faint objects. While this increase in data provides a boon to the task of asteroid detection, it vastly increases the complexity and scale of the problem.

This work deals with the computational issues inherent to large-scale target tracking and the development of techniques and algorithms that mitigate or eliminate these issues. The computational issues, which are discussed in detail in Section 2, include a variety of spatial and data association queries. Figure 1 illustrates one representative problem, track initiation. Observations from all of the time steps are shown on a single image with observations from different time steps represented as different shapes. The goal is to take the raw data (Figure 1.A) and find sets of observations that follow the desired linear motion model (Figure 1.B). The types of difficulties illustrated, such as sheer combinatorics of an exhaustive search, are typical of many of the tasks we will be examining.

While many of the queries and algorithms I will discuss are important in a range of fields, such as target tracking, computer vision, and computer graphics, our primary motivating example is the asteroid linkage and tracking problem. Here we wish to tractably identify moving objects and either attribute them to a known asteroid or link them with other observations in order to determine tentative orbits.

There are two primary factors that set asteroid tracking apart as an interesting and computationally challenging domain. The first, and perhaps most significant, is the

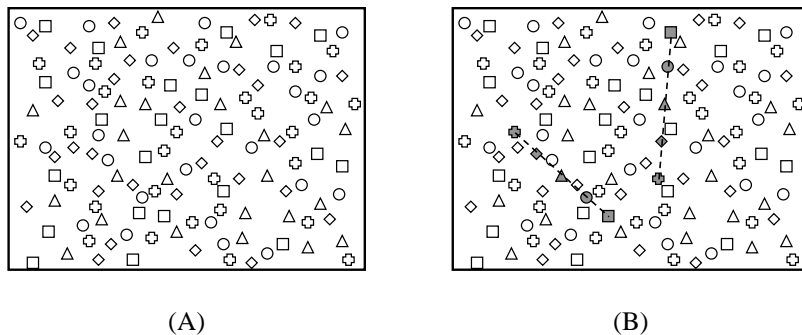


Figure 1: The linkage problem is to find one point at each time step such that the points fit the model for a candidate track. Points from each of the five different time steps are shown as different shapes (square \rightarrow circle \rightarrow triangle \rightarrow diamond \rightarrow plus). Two linear linkages are shown (B) and a third is left as an exercise for the reader.

sheer scale of the problem. New sky surveys give the potential to track hundreds of thousands of objects. Further, the astronomy domain has the unique property that we can push down into the noise to track increasingly faint objects by decreasing the brightness threshold required to consider a detection. However, as we examine fainter detections we increase the number of true objects and are more susceptible to false detections arising from noise. In fact, the number of detections per image increases faster than exponentially as the signal to noise ratio decreases [17]. The second factor is the observational schedule. Even under ideal conditions each region of the sky will be observed only relatively infrequently. Detections of a given object may be sparse, sporadic, and widely spaced. Each individual viewing will only cover a small fraction of the sky, leaving the majority of the objects unobserved at a given time step. Thus we must piece together these sparse pieces of evidence to extract the true tracks. Finally, additional complications, such as the fact we only observe a subset of an asteroid's position information, further increase the complexity of the task.

The rest of the document is organized as follows. In Section 2 I provide a brief overview of the computational problems that arise and outline which aspects I will be investigating. In Section 3 I provide a brief overview of the specific domain of interest, asteroid tracking. Then I introduce several pieces of current and completed work. In Section 4, I discuss the first version of the asteroid linkage algorithm. In Section 5 I discuss the novel use of multiple trees to accelerate track initiation and asteroid linkages. In Section 6 I will discuss the use of spatial data structures to accelerate occurrence, attribution, and precovery queries. Finally, in Section 7 I outline the contributions I plan to make in this thesis and provide a timeline.

2 Problem Overview

At its core, the tracking problem is the determination of which observations were generated by the same object and the estimation of this object's underlying state or trajectory. In the case of asteroid tracking we are specifically interested in determining which observations in the night sky correspond to the same asteroid and estimating that asteroid's orbit. To this end we are interested in such problems as:

- *Linking* together observations that fit an underlying movement model so as to find and initiate new tracks. This process is also called *track initiation*.
- *Predicting* where a potential track will be at a future time so that we may search for supporting observations.
- *Associating* estimated tracks with new observations.
- *Incorporating evidence and maintaining* the set of potential tracks so as to provide an accurate set of track estimates.
- *Precovering* and/or *attributing* old observations to new track estimates. Once we have a tentative track/orbit we can go back and find observations corresponding to this track in previous time steps or even other databases.

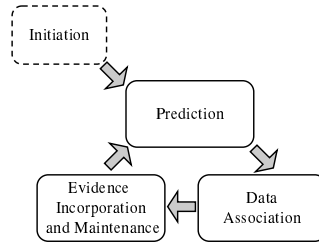


Figure 2: The flow of steps for basic sequential target tracking.

Below I discuss these aspects in more detail, outlining which ones I will cover in this thesis.

These aspects correspond to components of the general multiple target tracking problem. A common approach to target tracking is to treat the problem sequentially, applying these aspects repeatedly at each time step. Figure 2 illustrates this process. In the first step, the current information about the targets is used to predict them ahead to the subsequent time step. The predicted positions are then used to associate the tracks with new observations. In the final step, the associations are used to refine the information about each target. Track initiation may be used to start this process by finding initial tracks and estimating their parameters. For a good introduction to general target tracking see [2] or [3].

2.1 Computational Challenges and Contributions

2.1.1 Initiation

The track initiation or linkage problem consists of taking sets of observations from different time steps and *linking* together those observations that fit a desired model *without* initial track estimates. Thus the goal is to find and estimate candidate tracks from the raw observation data. This problem is the core of the asteroid linkage problem; we want to find candidate associations given sparse and noisy sets of observations.

In this thesis, the problem of track initiation will be the primary area of investigation. Specifically, I will be looking at cases where the data may contain a high number of both true objects and noise observations, but is temporally-sparse, containing relatively few observations of the same region spaced over a wide range of time. This problem is especially difficult because the high number of points initially gives rise to many short, spurious, candidate tracks and the temporal sparsity means that a significant time window may be necessary to fully confirm a new track.

2.1.2 Prediction

The prediction problem consists of estimating the future positions of current tracks. Given a full estimated track model, this step consists of simply evaluating that model at the new time. Although prediction may not be the most costly step, it is a vital component of a tracking system. It is also very closely related to the spatial data association step.

In this thesis I will be examining the possibility of using spatial data structures to improve the performance of the prediction step. More importantly I will be examining algorithms that attempt to combine the spatial structure in the prediction and data association steps to improve performance and reduce computational cost. Further, by the nature of the asteroid linkage problem, I will be considering cases where a track must be predicted to a later time step *without* a full estimate of its parameters.

2.1.3 Data Association

The data association problem consists of determining which tracks correspond to which observations. Data association can be broken into two separate steps: *spatial* and *combinatorial* data association. In the first step, all of the potential associations are identified. This problem is usually equivalent to finding observations that are “close” to the predicted track position given the model’s error information. This step provides a list of potential track/observation pairings and may include multiple pairings containing the same track or the same observation. In the second step, these potential associations are resolved by assigning the observations to tracks.

In this thesis I will be considering the spatial problem of tractably identifying potential associations. However, I will not be considering the problem of combinatorial data association. There are many different approaches for this later problem covering a wide range in accuracy and computational cost. I plan to use a subset of these established approaches.

2.1.4 Evidence Incorporation and Maintenance

The evidence incorporation and maintenance steps provide updated estimates of the current set of tracks. The evidence incorporation step updates the estimate of each track using the new information provided by its associated observation. The maintenance step updates the set of tracks by confirming new tracks and pruning out dead tracks.

There exists many well established and tested techniques for both components [3], such as Kalman Filters, particle filters, and multiple hypothesis tracking. In this thesis, I do not plan to investigate new approaches for evidence incorporation and maintenance. Instead I plan to use established approaches.

2.1.5 Trajectory-Based Spatial Queries

Trajectory-based spatial queries deal with the question of whether a region of space is “close” to the trajectory of a given moving object. For example, we may wish to determine which past observations are near a newly discovered track. These queries are described in more detail in Section 6. While these queries do not fall into the basic

tracking description, they are vital for the asteroid linkage problem. We want to be able to find past observations of a newly discovered object in order to confirm its existence and refine its orbit.

In this thesis I will be examining trajectory-based spatial queries. Specifically, I will be examining the use of spatial data structures to accelerate these queries on large sets of both tracks and observations. Further, I plan to investigate the use of these queries within a tracking algorithm for such tasks as calculating support for a track to facilitate early pruning.

2.2 Notation

Tracks. At the heart of the problem we are interested in estimating tracks corresponding to both the association of a set of observations and the trajectory of a true underlying target. We allow a general definition of a track as any function of the independent variable through the D dimensional space. We denote the i th track as $\mathbf{g}_i(t)$ and use N_T to denote the number of tracks. Our discussion below focuses primarily on two types of tracks: linear and quadratic. The quadratic track is simply a quadratic function of time:

$$\mathbf{g}(t) = \mathbf{a} \cdot t^2 + \mathbf{b} \cdot t + \mathbf{c} \quad (1)$$

and can be used to describe physical motions of objects undergoing constant acceleration. The linear track is a linear function of time:

$$\mathbf{g}(t) = \mathbf{b} \cdot t + \mathbf{c} \quad (2)$$

and can be used to describe the physical motion of objects traveling at a constant velocity. In addition, the linear model can be used for such queries as finding lines or edges described by the observations. While much of our discussion and techniques presented below will also apply to other track models, we restrict the discussion to the linear and quadratic models to keep the discussion simple and consistent.

Observations. The observations take the form of D dimensional points. We denote the i th observation as \mathbf{x}_i and indicate the time of observation as t_i . We use N_X to denote the number of observations.

Plates. In many queries we consider entire viewing regions of observations, called *plates*. Below we consider rectangular plates where each plate W contains: the time of observation t , a vector indicating the upper bounds of the plate \mathbf{h} , and a vector indicating the lower bounds of the plate \mathbf{l} . While we restrict our discussion to rectangular plates for simplicity, many of the approaches will apply to plates of other shapes. We use N_P to denote the number of plates.

Linkage. A linkage is a set observations. We are interested in finding valid linkages: sets of observations that correspond to our given feasibility criteria for being a valid track.

3 The Asteroid Domain

As described in the introduction, the scale and sparsity of observations make asteroid tracking a computationally interesting problem. Below I provide a brief overview of

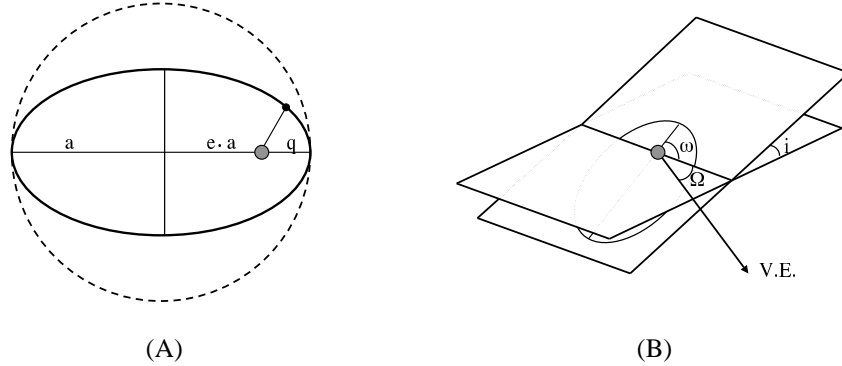


Figure 3: An orbit is uniquely defined by six parameters: three (a, e, t_0) describe the orbit on a plane (A) and three (i, Ω, ω) describe the orientation of the orbit (B).

the asteroid tracking domain, including: a simple model of asteroid motion and orbits, the requirements of a linkage system, and previous approaches to the asteroid linkage problem. The information in Sections 3.1 is drawn largely from [5] and [15]. Finally, I conclude by examining why increasingly robust and efficient algorithms are necessary in this domain.

3.1 Brief Overview of Asteroid Orbits

At first glance, the motion of an asteroid appears to follow a very simple model. For the most part the motion of an asteroid is dominated by its interaction with the sun. In this simple two body system, an asteroid orbits the sun according to Kepler's laws. Primarily, the orbit is confined to a single plane and traces out a conic section on this plane. Further, this motion is constrained by the law of equal areas, which states that the asteroid's radius vector sweeps out equal areas of the conic section during equal time periods of its orbit.

An orbit can be fully specified by six parameters: three specifying the orbit's conic section on its orbital plane and three determining the orientation of the orbit. Figure 3 illustrates these parameters. Figure 3.A shows an elliptical orbit on the orbital plane. This orbit is defined by the width of the orbit (semi-major axis a or perihelion distance q), eccentricity (e), and the time when the object passes closest to the sun (time of perihelion t_0). Figure 3.B shows the orientation of the orbit relative to earth's orbital plane, which is defined by three angles of rotation (i, Ω, ω).

Despite this simple motion model, an asteroid's movement across the sky may be significantly more complex. The actual observation is a combination of the object's movement around the sun and the earth's movement around the sun. Both of the orbits are more complicated than the simple two body problem presented above as they are subjected to perturbations from other objects, such as Jupiter and the Earth's own moon.

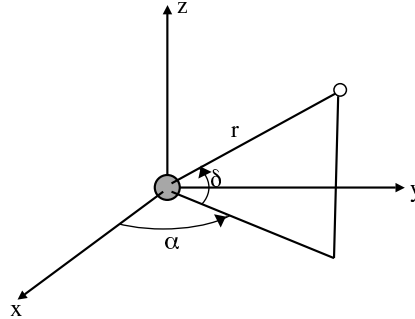


Figure 4: The Right Ascension/declination coordinates for an observation. The x and y axis lie along the Earth's equator and the z axis points along the Earth's north pole.

One of the major difficulties in asteroid tracking and orbit determination arises from the fact that we only observe a subset of the object's position information. As shown in Figure 4, the position of the asteroid relative to the earth at any given point of time can be defined by three spherical coordinates: Right Ascension α , declination δ , and distance r . Telescope observations only provide the two angular coordinates (α, δ) and we do *not* see the distance r . For this reason our initial work focuses on tracking in 2-dimensional observation space.

3.2 Asteroid Linkage Requirements

The key requirement of an asteroid linkage algorithm is to collect a sufficient set of observations that can be used to estimate an orbit. In this respect, the problem of asteroid linkage corresponds largely to the problem of track initiation. At the minimum, we need two *full* observations (α, δ, r) to determine a tentative orbit [5, 15]. Unfortunately, in most cases we do not have the distance to an object and must fit the orbit with only the angular coordinates. In this case, we need at least three observations [5, 15].

For brevity of this proposal, I do not describe orbit fitting algorithms, but rather mention several important trends. First, we need multiple observations in order to fit an orbit. Second, in order to accurately estimate an orbit, these observations need to be "reasonably" well spaced in time [14]. The necessary spacing in time depends on such factors as the curvature of the orbit and the accuracy of the observations, but we would like to develop algorithms with the capability to link observations over at least a few weeks. For example, Marsden [13] states that making additional observations "such that the total arc is less than five days or so" is "invariably a waste of time". Both of these requirements correspond to the need for accurate asteroid linkage algorithms.

3.3 Previous Approaches

Currently the most common approach to finding new asteroids is to look for sets of observations with a roughly linear movement over a short time span [10, 14]. Several closely spaced observations are taken each night and used to estimate the object’s linear movement across the sky [13, 14]. These closely spaced observations can be accurately linked together by their immediate proximity into an *attributable* containing an estimated position and angular velocity [14]. The linear parameters are then used to project the asteroid to later nights where they are associated with other observations. Finally, the proposed sets of observations are tested by examining their proximity to the best fitting orbit. The use of orbit fitting provides a tight and accurate pruning criteria to reduce the number of false linkages.

To our knowledge this process has been restricted to *linear* projections over short time spans and does *not* use spatial structure in the data to reduce the computational cost. This approach has several significant drawbacks. First, as the number of observations increases, the computational cost of finding candidate associations increases proportionally. Second, the use of linear projections is a very rough approximation of the true orbital motion and will only be valid over a very short time span. Finally, while the orbit fitting provides a tight and accurate pruning criteria, it is only applied after sets of observations have been found using linear approximation. Thus it is possible that we may need to fit orbits to *many* incorrect sets and that true linkages may be falsely rejected by the linear approximation.

3.4 The Need for Better Linkage Algorithms

Above we presented several reasons why the asteroid linkage problem is a computationally interesting one. Another equally important question is whether there actually exists a need in the astronomy community for more efficient algorithms. Two potential arguments can be ventured against this need. First, observations can only be collected during the night, leaving significant time for processing and tracking during the day. Further, there are periods when observing becomes undesirable or even impossible, such as poor weather conditions and nights with a near-full moon. These time periods could also be used to handle computational intensive algorithms. Second, the initial linkage problem promises to get *less* computationally intensive over time. As more asteroids are found, their corresponding observations can be identified in future images and there will be fewer unknown observations.

The first argument neglects potential expansions and/or needs of the asteroid tracking problem. The use of multiple observatories (including space-based telescopes) may effectively remove observation “downtimes.” Further, this argument neglects the possible need for “online” tracking. Many dim asteroids may only be visible for short periods of time when they are brightest, such as when they are at opposition [13]. We may wish to identify these new objects quickly in order to obtain additional observations before the objects are no longer visible. Finally, as we begin to track dimmer objects, the combinatorics of the problems increase significantly. In fact, the number of detections per image increases faster than exponentially as the signal to noise ratio decreases [17]. Thus such gaps in time may no longer be enough for inefficient

algorithms.

The second argument illustrates a fundamental goal of asteroid tracking, to find and track *all* sufficiently large objects. While this goal limits the sustained usefulness of algorithms that are specific for the discovery of new asteroids, it makes such algorithms *more* important in the short term. By finding and tracking the bright objects in the sky, we can focus the search on pushing down into the noise to track increasingly dim objects. As we shift the focus to dimmer objects, the need for efficient algorithms becomes greater. Further, many of the algorithms and approaches we are developing will be useful for future long term needs of the asteroid tracking problem. For example, as the number of known objects increases, the need for efficient algorithms for such tasks as attributing an observation to a known orbit and orbit matching *increase*. By developing techniques that aid both the discovery and maintenance of asteroid tracks, we can aid the tractability of asteroid tracking in both the short and long term.

4 Current and Completed Work: Initial Asteroid Linkage Algorithm

Our first approach at an efficient asteroid linkage algorithm was to bring in and use methods from the general field of tracking to improve the accuracy and reduce the computational cost for asteroid linkage. Specifically, we used a simplified version of multiple hypothesis tracking with a more flexible quadratic motion model and more efficient data association via spatial data structures.

4.1 Simplified Multiple Hypothesis Tracking

We use a sequential tracking approach to detect possible linkages (for a good introduction to sequential tracking algorithms see [3] and references therein). Each attributable (set of closely spaced and linked observations) is considered in turn as the start of a new tentative track. This tentative track is then projected to subsequent time steps and associated with later observations to confirm the track and refined the estimated track parameters.

In order to accommodate multiple possible track/observation associations at each time step, we use a simplified version of multiple hypothesis tracking. When a tentative track matches multiple observations at a given time step, multiple hypotheses (tentative tracks) are formed and the decision of which association is correct is delayed to a later time step. This process is illustrated in Figure 5. The single point matches three other points at the second time step. These points are used to create three hypothesized tracks. This process continues to the third and fourth time step with “bad” hypotheses being pruned away.

In order to reduce the number of candidate neighbors examined, *gating* is used. As shown in Figure 6, neighbors are first filtered by whether they fall within a window or gate around the track’s predicted position. This approach has also been used in conjunction with kd-tree structures to quickly retrieve the candidate observations near the predicted position of a track [21, 22].

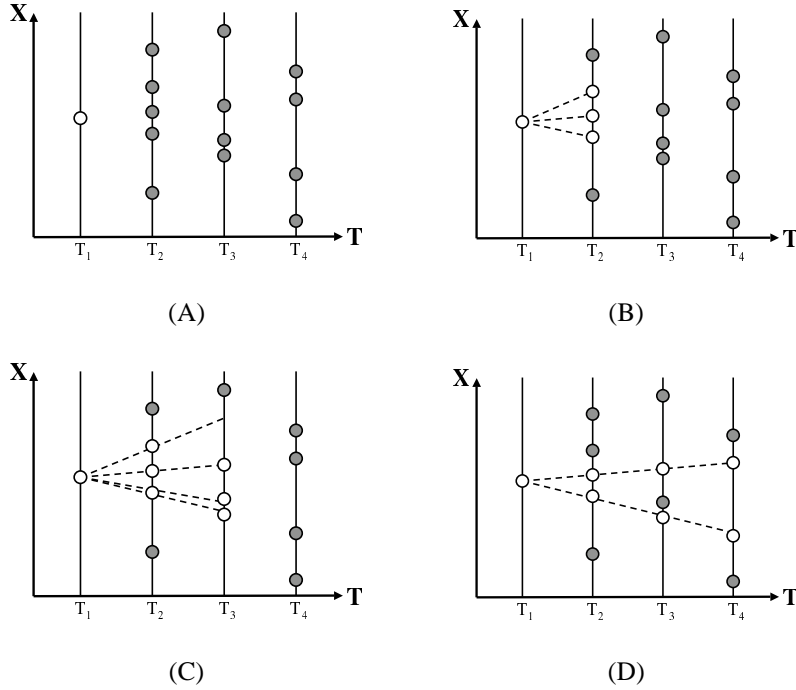


Figure 5: A multi-hypothesis tracker starts from a tentative track (A) and sequentially checks the later time steps. If multiple points fit a candidate track then several hypothesis are created (B) and (C).

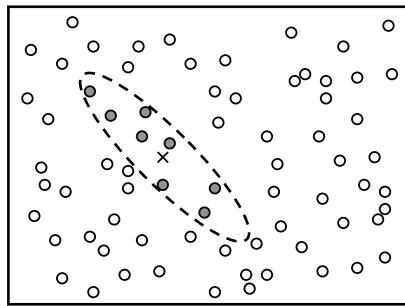


Figure 6: Gating can be used to ignore points that could not be part of the current track. The predicted position of the track is shown as an X and the points that fall within the gate are shaded.

It is important to note that our algorithm does not do any combinatorial data association. *All* sets of observations that fit the quadratic motion model are returned so that

they can be further filtered using orbit fitting. Because we do not use combinatorial data association, the tracks returned will not necessarily be disjoint. We do a limited amount of filtering by removing some tracks with a high degree of overlap. Specifically, we remove returned tracks that are subsets of other returned tracks and merge two tracks if they are compatible and contain over 50% overlap. Compatibility is defined as not containing conflicting observations (two different observations at the same time) and a good fit to the resulting merged track.

4.2 The Use of Quadratic Tracks

In order to improve the accuracy of the tracking algorithm, we used a quadratic motion model. An observation's position in α and δ was modeled as two independent quadratic functions of time. This model provided a better fit of an asteroid's movement (over the approximately 2 week time) and thus allowed us to more tightly prune sets of observations.

Unfortunately, the initial paired observations only contain estimated position and velocity information. Thus we used two different types of prediction. If the track contained enough information to estimate a quadratic model, we used quadratic prediction. Otherwise we used linear prediction.

4.3 The Use of Spatial Structure

In order to make the algorithm tractable on large sets of observations, we used kd-trees to accelerate spatial data association [21, 22]. Specifically, we built one kd-tree on the observations from each time step, which corresponded to a single image of the night sky. These kd-trees were then used to accelerate spatial data association. The estimated tracks were projected to the new time step and the predicted positions were used as queries for a spatial range search. For each hypothesized track, we efficiently found all nearby observations and used these to create new hypothesized tracks.

4.4 Results

To test the major components of the above approach, we applied the algorithm to simulated asteroid data with and without the various improvements. Specifically, we used the simplified multiple hypothesis tracker both with and without a kd-tree and with both linear and quadratic track approximations.

The data was generated from simulated orbits of approximately 1,000,000 main belt asteroids by calculating the object's position (Right Ascension and declination) at 8 different times: twice a night on every fourth night. The two observations from the same night were treated as given attributable. These observations were then filtered to only those that appeared in a given 100 square-degree region. For the test we used only the first $\frac{1}{10}$ th of the observations. In total we examined 20428 observations from 3320 different asteroids.

The performance of the algorithms was monitored by several different measures:

1. *Running Time* - The wall clock running time.

+	+	+	+	+	+	+	+	+	+	-	-
0	0	0	0	0	0	0	0	0	0	1	3

+	+	+	+	+	+	-	+	+	+	-
0	0	0	0	0	0	1	2	3	4	6

+	-	+	-	+	+	+	+	+	+	+	
0	1	2	4	6	8	10	12	14	16	18	20

Figure 7: The calculation of weighted number incorrect for three different results. The ordered correct (+) and incorrect (-) tracks are shown with the corresponding scores underneath. The triangles mark the 90% correct seen point and thus the final score. This score accounts for both the number and position of the false positives.

2. *Percent Correct* (P_C) - The percentage of returned tracks that exactly matched at least one true underlying track. This score measured the false positive rate.
3. *Percent Found* (P_F) - The percentage of true underlying tracks that exactly matched at least one of the returned tracks. This score measured the false negatives.
4. *Weighted Number Incorrect* (WI) - The weighted number of *incorrect* tracks (false positives) that have a better score than at least 10% of the true tracks. The weight of each track is determined by the number of true tracks the incorrect track's score exceeds. This score was calculated by sorting the returned tracks according to their fit to the given model and traversing the list from best to worst. At each step, the total score is incremented by the number of incorrect tracks seen so far. The counting stops after 90% of the true tracks have been seen. This calculation is illustrated in Figure 7. This score accounts for both the number and position of false positives.

It is important to note that we can often directly trade-off percentage correct and percentage found by varying the acceptance and pruning thresholds. In order to compensate for this, we set the thresholds under all variations to produce roughly the same percent found.

The results are shown in Table 1. Using the quadratic approximation allows significantly tighter pruning bounds without increasing the number of false negatives. This in turns leads to less false positives and reduces computation. The use of kd-trees allows us to exploit the spatial structure and significantly reduce computation. It is important to note that the results are on data that is $\frac{1}{10}$ th of the desired density. The loose bounds for the linear approximation lead to a very large (and effectively intractable) explosion of tentative tracks on data of the full density. However, using quadratic tracks with kd-trees on the full density data allows us to find 96.97% of the asteroids in under 3 minutes. Thus the algorithm provides substantial improvements over previous brute force linear approaches.

Approximation	kd-Trees?	Time (sec)	P_F	P_C	WI
Linear	NO	93	0.9622	0.0206	832369763
Linear	YES	6	0.9622	0.0206	832369763
Quadratic	NO	59	0.9638	0.8867	55038
Quadratic	YES	3	0.9638	0.8867	55038

Table 1: All of the performance measures (running time, percent found, percent correct, and weighted number incorrect) show that the use of a tighter approximation and spatial data structures can significantly improve performance in asteroid linkage.

4.5 Discussion and Future Directions

The above approach is both simple and effective. However, there are several issues that must still be addressed to provide a more efficient and robust solution as we begin to track increasingly faint objects with more realistic observation schedules.

1. The initial linear prediction step may not provide a tight enough criteria for pruning “bad pairs” as the gap in time between viewings increases or the density of observations increases. This may lead to a combinatorial explosion in the number of bad initial pairings that must be considered.
2. The use of kd-trees at each time step is not well suited to realistic observation schedules where we would expect to have only a small subset of the asteroids observed at each time step.
3. The use of a quadratic approximation of track movement may not be sufficient to tightly and accurately prune out tentative tracks as the density of observations or noise points increases.
4. The first few associations (i.e. the initiation stage) of the algorithm uses a relatively weak pruning criteria that does not take into account information from future time steps or real-world physical constraints. Again this may lead to a combinatorial explosion in the number of bad initial pairings that must be considered.

These issues provide some of the main motivation behind the work described below and the proposed research described in Section 7.

5 Current and Completed Work: Efficient Track Initiation

The track initiation problem consists of taking sets of observations from different time steps and *linking* together those observations that fit a desired model *without* any initial estimates of the track parameters. This problem, illustrated in Figure 8, is fundamental to discovering new asteroids. Observing the same object at multiple times allows us to

begin to track the object and even to determine whether an observation is an object at all or is just noise. Further, in order to accurately determine an orbit of a new asteroid, we must first find observations of this object over a sufficient range of time.

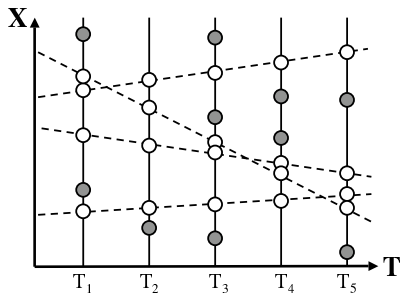


Figure 8: A set of one dimensional observations linked together by linear tracks. The white circles are the observations that correspond to the linear tracks (dashed lines).

Below I discuss previous approaches to this problem from the field of target tracking and a new multiple tree algorithm for exhaustively finding all feasible sets of observations. For brevity of this proposal, I only provide a brief outline of our approach to using multiple trees to accelerate exhaustive track initiation. Further details and additional experiments can be found in [11].

5.1 Problem Definition

Formally the linkage problem can be phrased as a filtering problem. At each time step k we observe N_k points from both the underlying set of tracks and noise. Given a set of observations at K distinct time steps, we want to return all tuples of observations such that:

1. the tuple contains exactly one observation per time step, and
2. it is possible for a single track to exist that passes within given thresholds of each observation.

Thus we wish to filter the $\prod_{k=1}^K N_k$ possible tuples down to just those tuples that could be feasible tracks.

The second condition specifies a constraint on the observations' fit to the underlying model. We say that a tuple of observations $(\mathbf{x}_{I_1}, \dots, \mathbf{x}_{I_K})$ is valid only if there exists a track \mathbf{g} such that each observation falls within some error bounds $[\zeta^L, \zeta^H]$ of the track in each dimension:

$$\zeta^L[d] \leq \mathbf{x}_i[d] - \mathbf{g}(t_i)[d] \leq \zeta^H[d] \quad \forall d, i \quad (3)$$

The thresholds ζ^L and ζ^H provide lower and upper bounds on the fit. This definition of feasibility is compatible with a range of statistical noise models. For example, we

can define an arbitrary observation noise model for the points on a track and set the thresholds in each dimension to be the 95% confidence interval for the noise in this dimension. Figure 9 shows an example of this. Further, we can vary ζ^L and ζ^H to account for systematic errors, time varying errors, and increased or decreased confidence in a track's estimate as new observations arrive.

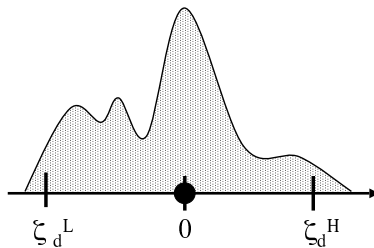


Figure 9: An arbitrary probability distribution and the resulting bounds. The circle denotes the observed location and the upper and lower bars indicate the acceptable locations for the track.

Our feasibility criterion differs slightly from many traditional filtering algorithms. Instead of asking about a new observation's fit to the current track estimate, we are asking an existence question. *Does there exist any set of parameters that will satisfy the given observational noise bounds?* Thus we do not need to worry about uncertainty in the track parameters during a prediction step. Instead we are re-asking the existence question for the entire new set of observations, the previous track plus the new candidate observation.

In contrast to the flexibility for noise models, it should be noted that the above criteria does not allow for a concept known as *process noise*. This means that we assume the track *always* follows the model. For example, a linear track model cannot account for changes in velocity. However, for the task of track initiation we are only dealing with short time ranges and thus with minimal process noise.

5.2 Previous Approaches

There are a variety of different approaches to the problem of track initiation. Below I briefly discuss some of the more common ones. These approaches differ from our own in several important ways. First, we are asking a different type of query. Specifically, we are asking for *all* sets of observations that could feasibly belong to a path. Second, we provide an exact algorithm for answering this query.

5.2.1 Sequential Tracking and Track Initiation

One common approach to track initiation is a sequential track initiation algorithm such as the one described in Section 4.1. The unassociated points are treated as new tracks and projected to the later time steps where they are associated with other points to form longer tracks. As described in Section 4.1 we can use a simple form of multiple hypothesis tracking to account for multiple possible track/observation associations. Further, we can use gating in conjunction with kd-trees to efficiently find candidate associations.

There are several potential disadvantages of this type of approach that arise from the sequential nature of the search itself. It does not use evidence from later time steps to aid early decisions. Many early good looking pairs may be easily pruned using a lack of further points along the track. Further, this approach has the potential of being thrown off by noise early in the track. Multiple hypothesis tracking attempts to mitigate this problem by allowing multiple tentative tracks, but introduces another problem, the possibility of a high branching factor causing a significant computational load.

It should be noted that sequential track initiation has the advantage that it can be applied to multiple tracks simultaneously. This gives this approach the ability to discount observations that are “obviously” members of other tracks.

5.2.2 Parameter Space Methods

Another approach to track initiation is to define a space indexed by the track parameters and to search for tracks in this new space. In the case of one-dimensional linear tracks, this space would have two dimensions (slope and intercept). One such popular algorithm is the Hough transform [8]. The idea behind these approaches is that for many simple models, individual observations correspond to simple regions or curves in parameter space. For example, in the one-dimensional linear case an observation (t, x) corresponds to the line:

$$c = (-b)t + x \tag{4}$$

An example with a linear model is shown in Figure 10. The points are shown in Figure 10.A and their corresponding lines in parameter space in Figure 10.B. If a series of observations lie along a line $(x = bt + c)$, then their lines in parameter space will intersect at a common point (b, c) . The Hough transform looks for common points of intersection by using grid-based counts of the number of lines that go through a particular region of parameter space (Figure 10.C and 10.D).

There are several major downsides to the parameter space approach. First, maintaining and querying the parameter space representation can be expensive in terms of both computation and memory. There are many possible intersections to check and storing occurrences in a grid structure may require significant amounts of space. Second, the level of discretization of parameter space can drastically affect the accuracy of the algorithm. If the grid is too tight then a small amount of noise can cause intersections to spread out over several bins and be missed. If the grid is too loose then coincidental occurrences can accumulate and cause false alarms. Although the false alarms can be filtered out in post-processing, this step further increases the computa-

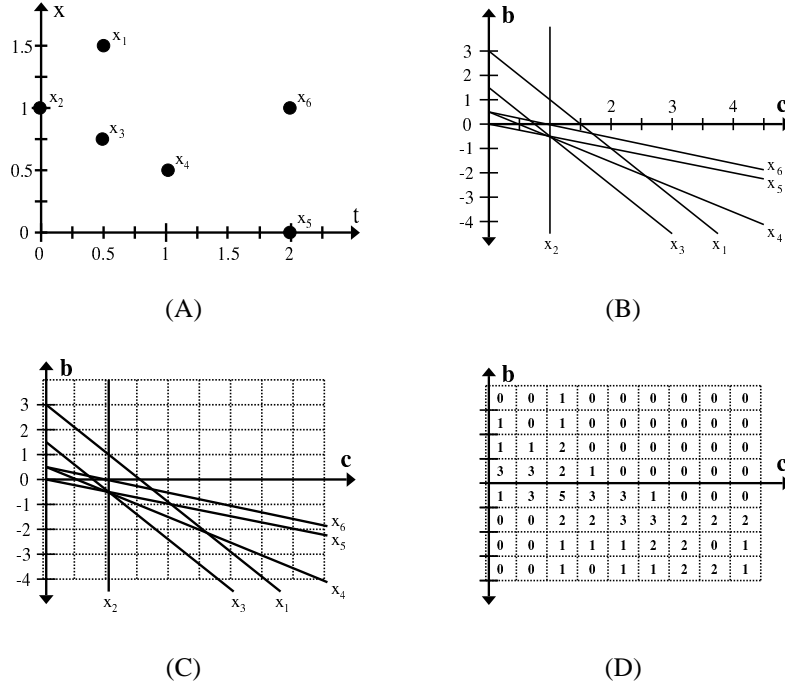


Figure 10: Under a linear model observations (A) correspond to lines in parameter space (B). If several observations lie along a line, their lines in parameter space will intersect at one place. Such clusters of intersections can be found by using grid based counts (C and D).

tional cost. Finally, in the presence of noise observations it may take many observations along the track to produce an obvious high density region in parameter space.

5.3 Multiple Tree Algorithm

Our solution to the problem of track initiation is to build *multiple* kd-trees over observations and traverse them simultaneously. Specifically, we build one tree for each time step that we wish to use. This approach allows us to not only look for pruning opportunities at the next time step, but also to consider pruning opportunities resulting from future time steps. Further, by considering multiple time steps together, we may be able to reduce redundant computation that can arise from pruning for similar points or initial tracks.

Figure 11 shows a one dimensional example of this approach. One tree is built independently on each of the K time steps. The algorithm starts at the root of each tree and begins a depth first search of combinations of tree nodes. At each level of the search the algorithm picks one node and recursively searches its children (Figure 11.B). When the search reaches a point where all K trees are at leaf nodes, the algorithm explicitly

tests all combinations of the points at these nodes and returns those tuples of points that fit the criteria for being potential tracks (Figure 11.C).

In the above form, the algorithm would search all combinations of the $\sum_{k=1}^K N_k$ leaf nodes, requiring $O(\prod_{k=1}^K N_k)$ time. The benefit of using the tree-based algorithm is that we can prune off a section of the search space if we can ever show that it is not possible to fit a track through the K nodes. We call such sets of nodes *infeasible*. Such a case is shown in Figure 11.D. This pruning criteria allows us to possibly ignore large numbers of the tuples that would have been tried under a brute force approach. We developed an efficient approach to determining whether a set of nodes is infeasible. A detailed description of this approach can be found in [11].

It should be noted that the use of multiple trees has been explored for quickly answering spatial queries [7]. However, this earlier work has so far been restricted to simple spatial proximity queries on points. In contrast, we consider the application of multiple trees to a more complex spatial problem with an inherent temporal component.

5.4 Sample Results

5.4.1 Algorithms

In examining the performance of the multiple tree approach, we compared the range of approaches from an adapted “sequential” approach to a full multiple tree approach. Specifically, we use multiple tree algorithms, denoted MT-1 through MT-K, with the following descent rule:

MT-k: If at least one of the first k trees is not at a leaf, descend the tree in the first k that owns the highest number of points. Otherwise descend the earliest tree that is not already at a leaf.

This rule runs a multiple tree algorithm on the first k trees and then confirms the potential tracks by sequentially examining the remainder of the tracks.

When $k = 1$, this algorithm mimics sequential track initiation. The algorithm descends the first tree until it reaches a leaf node. It then searches the second tree for points compatible with those in the first tree’s leaf node. The algorithm continues on in this manner, searching subsequent trees, and thus subsequent time steps, looking for points to confirm the tentative track. Pruning is only done in relation to whether the trees traversed so far allow a valid track. However, unlike many proposed sequential track initiation algorithms, this rule does not try to fit a track to the first few points and project this track ahead in time.

It is important to note that all versions of this descent rule (i.e. all values of k) use the same feasibility criteria. Thus all variations are exact algorithms and will return the same set of tracks.

5.4.2 Astronomy Data

One of the data sets that we examined was chosen to provide a realistic example of the asteroid linkage data. This allows us to ask whether we can accelerate the discovery of linkages on tracks of the volume, density and distribution of real asteroids. The

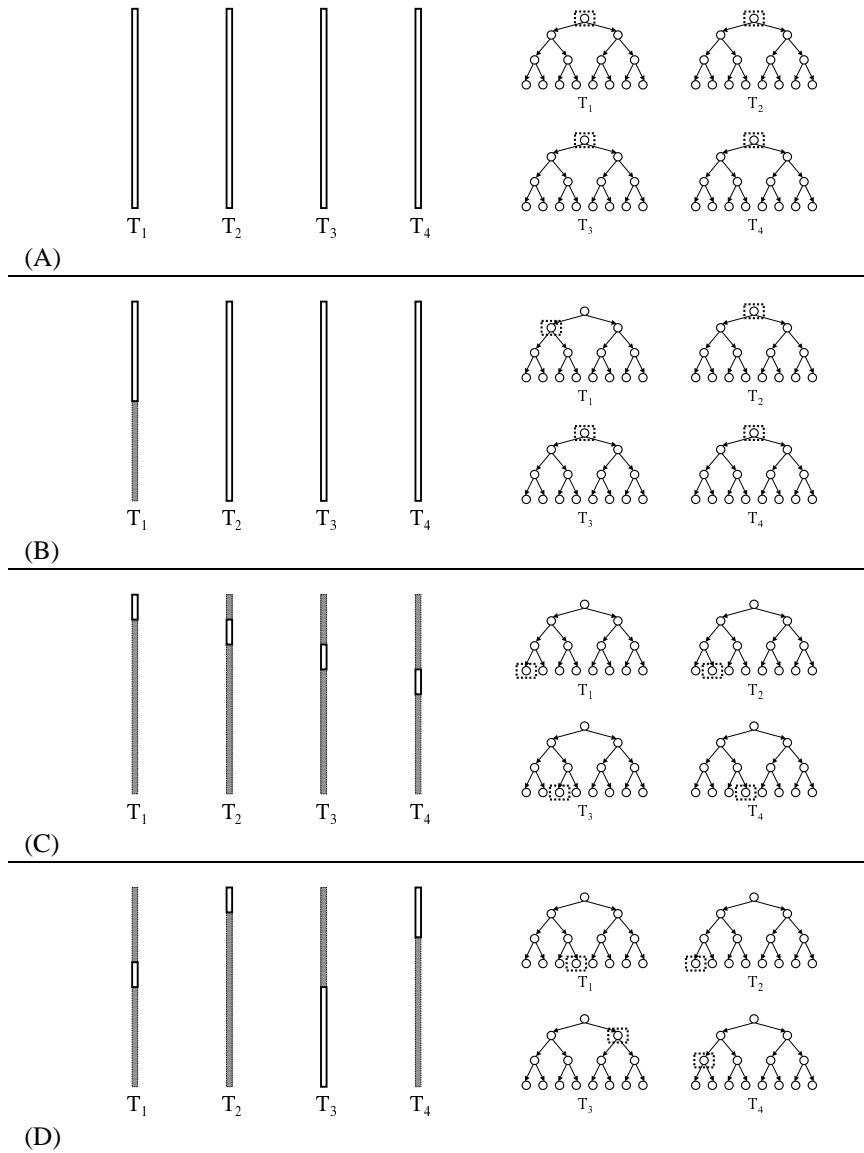


Figure 11: The multiple tree algorithm descends the trees in a depth first search (A and B). If it reaches the leaf nodes, it explicitly tests the tracks (C). The search can be pruned if it is not possible to fit a track through each of the nodes (D).

data consisted of simulated orbits for approximately 1,000,000 main belt asteroids and 1,800 near earth objects. These orbits were then approximated by a quadratic track over a period of 16 nights and observations were generated from this track. Each observation

T	MT-1	MT-2	MT-3	MT-4
4	232.38	232.30	213.40	108.10
5	69.19	69.11	58.17	38.77
6	28.93	28.86	21.91	19.68

Table 2: The number of pruning tests (in millions) for the astronomy data with varying numbers of observation and time step size.

consisted of two angular components, Right Ascension and declination, that gave the object’s location in the sky. These observations we generated on $T = 4$, $T = 5$, and $T = 6$ time steps equally spaced over the 16 nights. Finally, the observations were filtered to include only those appearing in a single 1 square degree region of the sky on at least one night, leaving a total of 1,768 different objects.

5.4.3 Results

Table 2 shows the results of this experiment and illustrates the potential benefits of a multiple tree approach. By using multiple trees we can significantly reduce the number of pruning queries needed. The differences between performance of different T is due to both a varying number of observations and the different spacing between observations.

Table 2 also points to another important trend. At the first few time steps we may have very little information, such as velocity or acceleration, about the track. Thus if the time until the next observation is large our predicted position may be significantly incorrect and we may have to check *many* neighbors at the subsequent time step. This problem is shown in Figure 12. The use of multiple trees mitigates this problem, by confirming early pairs with information from later time steps.

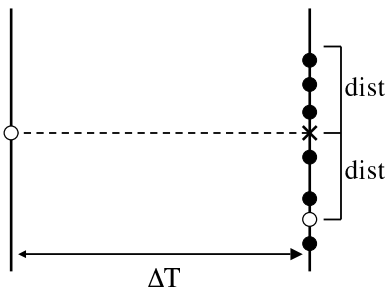


Figure 12: Without using velocity we would have to try at least 6 neighbors at the next time step to find the true next point along the track.

5.4.4 Discussion

The multi-tree method provides a tractable approach to exhaustive track initiation. One of the primary benefits of this new approach is that we can incorporate evidence from later time steps to aid in the pruning decisions of earlier time steps. Empirically, this algorithm performed very well on several simulated data sets, outperforming an exact adaptation of conventional multiple hypothesis tracking. Further details and additional experiments can be found in [11].

6 Current and Completed Work: Queries on Trajectories

Trajectory-based queries extend simple spatial queries to moving objects, asking about a track’s relation to points, regions, or other tracks. For example, we may wish to determine which observations fall close to a given track or which tracks pass close to a given observation. These types of problems are very important in asteroid tracking for answering such queries as:

- whether a track should appear in a given viewing region,
- which tracks could have generated a given observation, and
- which observations (in other time steps) may have been generated by a given track.

Below we consider several versions of this type of query and present several data structures and algorithms to efficiently solve these problems. The queries are tied together by their relation to the general query of which region/trajectory pairs are “close”, a similarity that allows us to develop algorithms that apply to all of the queries. Specifically, we rephrase all trajectory queries as questions of plate/track intersection. For brevity of this proposal, I only provide a brief outline of our algorithms to accelerate this type of query. Further details and additional experiments can be found in [12].

6.1 Query Types

6.1.1 Occurrence Queries

The occurrence query asks which tracks intersect which plates. This type of query is useful in determining whether a track *should* have an observation in a given region. Formally the occurrence or intersection problem can be phrased as a filtering problem. Given a set of plates and a set of tracks, we want to return all of the plate/track pairs, (W, \mathbf{g}) such that:

$$W.\mathbf{l}[d] \leq \mathbf{g}(t)[d] \leq W.\mathbf{h}[d] \quad \forall d \tag{5}$$

Thus we wish to filter the $N_T \cdot N_P$ possible pairs down to just the few that meet the above criteria.

6.1.2 Attribution Queries

The *attribution* query searches for tracks that pass “close to” a given point. Effectively, we are asking to which tracks we could possibly attribute a new observation. For this reason, the attribution query is very important in tracking and asteroid linkage for quickly associating new observations with known targets.

Formally, the attribution query can be specified as: given an observation \mathbf{x}_i return all tracks $\mathbf{g}(t)$ such that:

$$|\mathbf{g}(t_i)[d] - \mathbf{x}_i[d]| \leq \zeta[d] \quad \forall d \quad (6)$$

This query is a specification of the occurrence query. We can convert this query into an equivalent occurrence query by treating the observation as a small plate W :

$$\begin{aligned} W.t &= t_i \\ W.l[d] &= \mathbf{x}_i[d] - \zeta[d] \quad \forall d \\ W.h[d] &= \mathbf{x}_i[d] + \zeta[d] \quad \forall d \end{aligned} \quad (7)$$

Other distance measures can be used by adding a post-processing step. For any distance measure we can first approximate its threshold with a rectangle and then further filter the returned tracks using the true distance.

6.1.3 Precovery Query

The precovery query is the complement to the attribution query. Given a new track \mathbf{g} we wish to determine which points lie within some threshold ζ of this track. Formally, the precovery query can be specified as: given a track $\mathbf{g}(t)$ return all observations \mathbf{x} such that:

$$|\mathbf{g}(t)[d] - \mathbf{x}[d]| \leq \zeta[d] \quad \forall d \quad (8)$$

This question is important for such tasks as finding previous sightings of newly discovered asteroids. As with the attribution query, we can answer the precovery question by treating the observations as small plates and solving the occurrence query.

6.2 Previous Approaches

Previous approaches to accelerating queries of this type have been proposed in the fields of computer graphics, computer vision, moving object databases, and target tracking. Below I discuss the two predominant approaches: building tree-structures on the spatial regions and building tree structures on the tracks. In Section 6.3 I discuss adaptations of these approaches to the above queries in arbitrary domains.

The use of tree-based data structures on points or regions of space is a common approach to accelerate these types of spatial queries. In the problem of ray-tracing, placing objects in tree structures is a common and successful approach [6, 23]. In the field of tracking, tree structures are used to accelerate spatial queries for data association [22]. Here though, the data structures are built on the observations or predicted track positions *at each time step*. While this can be used to speed up data association

queries at a given time step, it is not applicable to our queries of interest. First, we are interested in cases where each time step may contain at most one observation. Thus building a tree on the plates or observations will not provide any benefit. Second, we are interested in plate or observation based queries, where we have many tracks but only a single plate.

The approach of building data structures on tracks has also been considered in a variety of domains. Arvo and Kirk proposed *ray classification*, a technique to accelerate computer ray tracing [1]. Rays are represented as points in 5-dimensional parameter space and partitioned into different groups. A similar technique has been presented in databases to answer queries about moving objects [9, 19, 16]. Again, the linear tracks are effectively treated as points in parameter space for the construction of a tree structure. By bounding the parameters, it is possible to create time parameterized bounds for the node in observation space. While this work has been restricted to linear models, it is important to note that the resulting trees are valid, but not optimal, for *all time*. We tested a similar method, extended to more complex track models, to provide comparison with the more efficient ball-tree based data structure. Finally, Pfoser *et. al.* presented two tree models for querying piecewise linear tracks [18]. These structures exploited the fact that 2-dimensional tracks could be broken into line segments and treated as a set of 3-dimensional objects.

6.3 Adaptations and New Algorithms

6.3.1 Brute Force Computation

Perhaps the simplest way to find all plate/track intersections is to exhaustively check each one. A brute force method runs a double loop through the data, comparing each plate with each track. This method is costly, requiring $O(N_P \cdot N_T)$ intersection tests. We implemented this algorithm for comparison purposes and denote it E in the following experiments.

6.3.2 KD-tree of Plates

One simple approach to speeding up spatial queries is to place the data in a spatial data structure such as a kd-tree. As described in Section 6.2 this type of approach has been used to accelerate ray tracing and spatial data association. In order to apply this approach to the above queries, we adapted it to work with nonlinear tracks in an arbitrary number of dimensions. Further because no two plates may occur at the same time, we may not gain any benefit from constructing a single tree at each time step. Instead we construct a *single* tree $D + 1$ dimensional tree, called a *plate tree*, on all plates by incorporating time as a dimension.

Our search for intersection then follows the same approach as a range search in a kd-tree. Given a query track $\mathbf{g}(t)$ we traverse the tree in a depth first search. If we hit a leaf node, we explicitly search the plates at this leaf for intersection using the criteria in Equation 5. Finally, we can prune the search if we ever find that the track cannot hit *any* plate contained within the bounds of the node. We can determine whether this criteria is met by taking the bounding box for the plate tree node (including time) and

asking whether the track intersects this box. If it does not, then we can safely prune the search.

We implemented the above adaption. This algorithm is denoted T_P in the following experiments.

6.3.3 KD-tree of Tracks

A natural complement to constructing a kd-tree on the plates is to construct one on the tracks. In many cases this approach might be a desirable alternative. For example, if we had a set of tracks and we wished to attribute a *single* new plate or observation to a track, we cannot achieve any acceleration using a plate tree.

One approach is to build a tree on the tracks by treating them as points in parameter space. For example, we can represent a D -dimensional quadratic track as $3D$ -dimensional vector by concatenating **a**, **b**, and **c** into a single vector. This approach has been proposed as *ray classification* to speed up ray tracing in computer graphics [1]. Here we adapt previous approaches by building a tree-structure on the resulting points. An example split is shown in Figure 13.

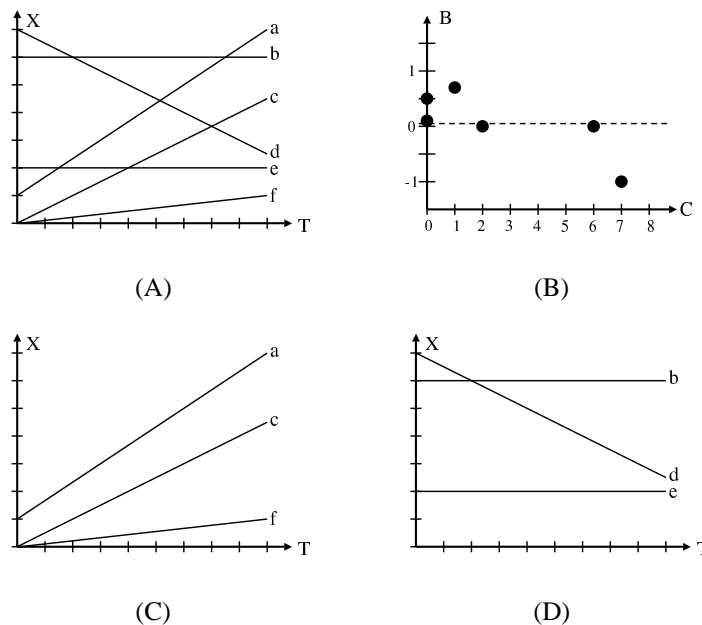


Figure 13: A set of linear tracks (A) is split using a dividing hyper-plane in parameter space (B). The resulting partition is shown in (C) and (D).

As with the plate tree, we store the bounding box for each node. However, this bounding box is on the parameters of the tracks. Therefore we are effectively storing the minimum and maximum parameters for each track owned by this node. We can use

these to calculate the bounds on where a track can be at a given time. For example, in the quadratic case:

$$\begin{aligned}\mathbf{g}_{MIN}(t)[d] &= \mathbf{a}_{MIN}[d] \cdot t^2 + \mathbf{b}_{MIN}[d] \cdot t + \mathbf{c}_{MIN}[d] \\ \mathbf{g}_{MAX}(t)[d] &= \mathbf{a}_{MAX}[d] \cdot t^2 + \mathbf{b}_{MAX}[d] \cdot t + \mathbf{c}_{MAX}[d]\end{aligned}\quad (9)$$

Our search for intersection then follows the same approach as for the plate tree. We do a depth first search of the tree using the bounding box of the node to look for pruning opportunities. Upon hitting a leaf we explicitly check the tracks contained at that leaf.

We call this data structure a track-based kd-tree and denote it T_K in the results. It is important to note that the implementation of this approach is provided for comparison.

6.3.4 Ball-tree of Tracks

An alternative approach to dividing the tracks by their parameters is to partition the tracks based on their “proximity” to other tracks in the set over the time of interest. The hope is that this partitioning will allow better splits to the data that both “splits” multiple parameters and takes into consideration the entire scope of time. We introduce one such tree, which is similar to a ball-tree or metric tree [4, 20]. We refer to this data structure as a track-based ball-tree and denote it T_B in the results.

The key idea behind the track-based ball-tree is a tight coupling between the pairwise fit between tracks and tree construction. Both the bounds for the nodes and the splits of the nodes during construction are determined by this distance measure. Specifically, the bounds of a node are defined with reference to a central *anchor* track, \mathbf{g}_a , and a node radius \mathbf{r} . The anchor track serves to indicate one possible predicted position of tracks in the node at a given time and the radius indicates the maximum offset from this prediction in each dimension. The radius is defined such that:

$$\left| \mathbf{g}_a(t)[d] - \mathbf{g}(t)[d] \right| \leq \mathbf{r}[d] \quad \forall t_s \leq t \leq t_e \quad \forall \mathbf{g} \in \text{node} \quad \forall d \quad (10)$$

where t_s and t_e indicate the time interval of interest. Thus $\mathbf{r}[d]$ is the furthest distance from the anchor in dimension d of any track owned by the node at any time in the range of interest. An illustration of these bounds is shown in Figure 14.

Again, the search for intersection follows the same approach. The major difference is in how we do the pruning query. A track in the node can hit a plate W if and only if the anchor track intersects a similar plate W' where:

$$\begin{aligned}W'.\mathbf{h}[d] &= W.\mathbf{h}[d] + \mathbf{r}[d] \\ W'.\mathbf{l}[d] &= W.\mathbf{l}[d] - \mathbf{r}[d]\end{aligned}\quad (11)$$

Checking for intersection between the anchor track and extended plate is the same as checking for intersection between the original plate and region within \mathbf{r} from the anchor’s position at that time.

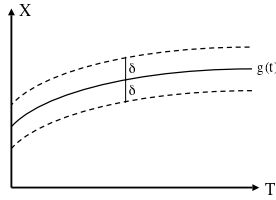


Figure 14: The track-based ball-tree nodes are defined in reference to an anchor track and radius. This radius indicates the maximum distance from the anchor track in each dimension.

6.3.5 Combining Trees

Since the above algorithms only build a structure on one component of the data, it is possible that a significant amount of work will be repeated over similar queries from the other portion of the data. This suggests a combined method that uses both a plate-based tree and a track-based tree.

We created a novel dual tree algorithm that uses both a plate tree and a track-based ball-tree. The idea behind that algorithm is that it does a depth first search of both trees at the same time. Formally, the search consists of two tree nodes (one from each tree) and at each level it recursively descends one of these trees. The “wider” tree is descended, moving the search down the tree with the least pruning power. If the search ever reaches leaf nodes in both trees it explicitly tries the track/plate intersections for those trees.

The true advantage of this search is that we can prune a subtree in the plate tree for all tracks in the subtree of the track-based tree. For data sets with many tracks and plates, this may provide a substantial advantage over only pruning subtrees of one component using individual members of the other component. We can safely prune if and only if we ever discover that it is not possible for *any* track owned by the current track node to hit *any* plate owned by the current plate node. This condition can be checked by asking whether the anchor track comes within its radius of the plate node’s bounding box. This pruning query is illustrated in Figure 15.

This dual tree method is denoted T_{PB} in the results.

6.4 Sample Results

Again we tested the algorithms using a data set that provided a realistic example of the asteroid data. The data consisted of simulated orbits for one million main belt asteroids and 1,769 near earth objects. These orbits were used to generate 8 observations with two observations per night on every fourth night. We then approximated the orbits as quadratic tracks and searched for all pairs that could be a valid attribution. We defined a valid attribution as any track/observation pair that fell within 0.001 in declination and 0.015 in Right Ascension of each other. These thresholds were chosen because they found at least 99.9% of the true attributions and at most 25% of the returned pairs were

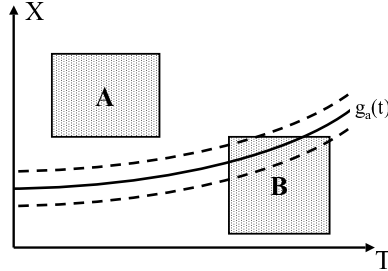


Figure 15: Pruning in the dual tree algorithm checks whether the track-based ball-tree’s anchor track is within the radius of the plate tree’s bounding box. Plate tree node A would be pruned while plate tree node B would not.

Size	N_T	N_X	E	T_P	T_K	T_B	T_{PB}
1	1768	8504	15.04	0.19	1.71	1.16	0.16
4	4395	28284	124.31	0.64	7.08	5.04	0.55
9	8731	57855	505.13	1.41	17.08	11.77	1.17
25	23066	160300	3697.48	4.23	43.28	38.21	3.31
100	66222	470477	31155.93	13.58	148.41	135.80	10.10

Table 3: Number of pruning queries (in millions) required to answer attribution queries for various sized regions (in square degrees).

incorrect. Finally, we varied the size of the region of sky under consideration from 1 square degree to 100 square degrees. Only observations and tracks that fell within this region were considered.

Table 3 shows the results of this experiment in terms of the average number of pruning queries. As we increased the size of the region, we included more tracks and observations and increased the size of the problem. All of the tree-based algorithms perform well, scaling significantly better than the exhaustive search. The high ratio of plates to tracks was an advantage for the plate tree algorithm. However, the consistent winner was the dual tree algorithm. It should be noted that while the number of pruning queries was chosen to provide a standardized measure that is independent of code optimizations such as data ordering and cache friendliness, the wall-clock running times of the algorithms also indicated approximately the same relative performance.

It should be noted that while the track-based tree algorithms did not perform as well as the plate tree algorithm, the track-based tree algorithms may often be the better choice in real-world applications. First, it may not be possible to restrict the tracks to the small set that correspond to the region of interest ahead of time. Therefore we may have significantly more tracks. Second, attribution queries may not always have all of the observation data up-front. If observations are coming in one at a time, then

it is better to use a track based algorithm. However, if the observations are coming in batches then a dual-tree algorithm is the best choice.

7 Research Plan and Future Work

7.1 Contributions

The primary contribution of this thesis will be the development of new approaches and algorithms for tractable tracking on domains with many targets and sparse or uncertain observation schedules. This work will differ from previous approaches primarily in scalability and robustness to sporadic observation schedules.

This work will focus on the following aspects of the tracking problem:

- *Initiation* - Efficiently and robustly identifying new candidate tracks from singleton observations and attributables.
- *Spatial Data Association* - Efficiently identifying potential track/observation associations using a combination of the known track information, any known model or noise information, and the spatial positions of the observations.
- *Prediction* - Efficiently predict a track's future position given the current information about the track.
- *Precoveries, Occurrences, and Attributions* - Efficiently answering "track-based" queries of whether a region of space is "close" to the trajectory of moving object.

Specifically, I will address the following research questions:

- *How can we further use spatial structure within the data to reduce the computational cost?* As shown above, the use of kd-trees can provide significant accelerations to the problem of spatial data association. It is likely that we can push the use of spatial data structures further for even more efficient algorithms.
- *Can we use (multiple) tree data structures to remove unnecessary or redundant computation in the above queries?* As shown in Section 5 and in [7], the use of multiple spatial data structures can improve performance by removing redundant computation that may result from processing similar queries. It is possible that this approach can be applied to other aspects of tracking and asteroid linkage to improve efficiency.
- *How can we remove assumptions about data arriving at fixed time steps to improve accuracy and reduce computation?* Many tracking algorithms make the simplifying assumption that data arrives at discrete time steps. In domains such as asteroid tracking we get small sets of observations spanning a range of times. It may be possible to exploit the (semi-)continuous nature of this data to improve accuracy and *reduce* computational cost.
- *How can we combine queries/operations so as to exploit additional structure within the data and reduce computational cost?*

- *How can we incorporate the physical constraints to further improve accuracy and efficiency?* Kepler's laws provide a significant amount of structure that is not currently being exploited. By incorporating this structure we may be able to increase the amount of pruning while reducing the number of true tracks that are incorrectly pruned.
- *How can we incorporate non-sequential tracking methods to survive high levels of temporal sparsity?* As the gaps between observation times increase, using sequential track initiation methods may become infeasible. More observations at the next time step will conform with potential velocity constraints and systematic errors from the approximate track models may become more significant. It may be possible to combine sequential tracking methods and non-sequential tracking methods, such as the Hough Transform [8], to reduce the computational cost of some queries.

These questions will provide potential techniques for making the queries tractable on large scale domains.

The algorithms are being developed with the goal of providing usable real-world software for our collaborators in the field of asteroid tracking. Thus a significant collaborative effort will be made to ensure that the software meets their requirements and provides tractable methods for asteroid linkages. This means that the algorithmic contributions of this thesis must be both applicable and deployable and will have immediate real-world applications.

7.2 Timeline

Below is my proposed timeline.

Spring 2005

- Investigate continuous time adaptation of sequential tracking algorithms. Specifically, examine the use of spatial data structures to preserve or extend current benefits resulting from the use of spatial structures at each time step while preserving the (semi-)continuous nature of the data.
- Investigate how widely spaced in time observations can be while retaining a high level of linkage performance.

Summer 2005

- Investigate "high noise density" track initiation (> 90% of observations are noise). Evaluate current algorithms and (if necessary) develop new approaches.
- Investigate the use of orbital information within the tracking and linkage algorithms.
- Investigate the possibility of performing the asteroid tracking in true 3-dimensional space instead of 2-dimensional observation space.

Fall 2005

- Investigate faint object track initiation. This means pushing down into the noise and finding increasingly faint objects.

Spring 2006

- Publish final version of tracking software for the astrophysicists.
- Finish written thesis and defend.

Acknowledgements

Jeremy Kubica is supported by a grant from the Fannie and John Hertz Foundation.

References

- [1] J. Arvo and D. Kirk. Fast ray tracing by ray classification. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, pages 55–64. ACM Press, 1987.
- [2] Y. Bar-Shalom, X. R. Li, and T. Kirubarajan. *Estimation with Applications to Tracking and Navigation: Theory Algorithms and Software*. Wiley-Interscience, 2001.
- [3] S. Blackman and R. Popoli. *Design and Analysis of Modern Tracking Systems*. Artech House, 1999.
- [4] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd VLDB International Conference*, September 1997.
- [5] P. R. Escobal. *Methods of Orbit Determination*. John Wiley and Sons, 1965.
- [6] A. S. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10):15–22, 1984.
- [7] A. Gray and A. Moore. N-body problems in statistical learning. In T. K. Leen and T. G. Dietterich, editors, *Advances in Neural Information Processing Systems*. MIT Press, 2001.
- [8] P. V. C. Hough. Machine analysis of bubble chamber pictures. In *International Conference on High Energy Accelerators and Instrumentation*. CERN, 1959.
- [9] G. Kollios, D. Gunopulos, and V. Tsotras. On indexing mobile objects. In *Proc. of the 18th ACM Symp. on Principles of Database Systems*, 1999.
- [10] L. K. Kristensen. Tracking faint asteroids. In *In Press - Preliminary Draft*, 2003.
- [11] J. Kubica, A. Moore, A. Connolly, and R. Jedicke. Fast and robust track initiation using multiple trees. In *CMU Tech. Report 04-62*, 2004.
- [12] J. Kubica, A. Moore, A. Connolly, and R. Jedicke. Spatial data structures for efficient trajectory-based queries. In *CMU Tech. Report 04-61*, 2004.
- [13] B. G. Marsden. The computation of orbits in indeterminate and uncertain cases. *Astronomical Journal*, 102:1539–1552, October 1991.
- [14] A. Milani, G. F. Gronchi, M. de’ Michieli Vitturi, and Z. Knezevic. Orbit determination with very short arcs: I admissible regions. *in press Celestial Mechanics*, 2004.
- [15] O. Montenbruck and T. Pfleger. *Astronomy on the Personal Computer*. Springer, 4th edition, 1999.
- [16] D. Papadopoulos, G. Kollios, D. Gunopulos, and V. Tsotras. Indexing mobile objects on the plane. In *MDDS*, 2002.
- [17] J. M. Petit, M. Holman, H. Scholl, J. Kavelaars, and B. Gladman. A highly automated moving object detection package. *Monthly Notices of the Royal Astronomical Society*, 347(2):471–480, September 2003.

- [18] D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel approaches to the indexing of moving object trajectories. In *Proc. 26th Int'l Conference on Very Large Databases*, September 2000.
- [19] S. Saltinis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. In *SIGMOD Conference*, pages 331–342, 2000.
- [20] J. K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40:175–179, 1991.
- [21] J. K. Uhlmann. Algorithms for multiple-target tracking. *American Scientist*, 80(2):128–141, 1992.
- [22] J. K. Uhlmann. Introduction to the algorithmics of data association in multiple-target tracking. In D. L. Hall and J. Llinas, editors, *Handbook of Multisensor Data Fusion*, pages 3.1–3.18. CRC Press, 2001.
- [23] A. Watt. *3D Computer Graphics*. Addison-Wesley, 3rd edition, 2000.