Fast and Robust Track Initiation Using Multiple Trees

Jeremy Kubica, Andrew Moore, Andrew Connolly, and Robert Jedicke

CMU-RI-TR-04-62

November 2004

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

In this paper we examine a fundamental problem in many tracking tasks: track initiation (also called linkage). This problem consists of taking sets of point observations from different time steps and *linking* together those observations that fit a desired model without any previous track estimates. In general this problem suffers from a combinatorial explosion in the number of potential tracks that must be evaluated.

We introduce a new methodology for track initiation that exhaustively considers all possible linkages. We then introduce an exact multiple kd-tree algorithm for tractably finding all of the linkages. We compare this approach to an adapted version of multiple hypothesis tracking using spatial data structures and show how the use of multiple trees can provide a significant benefit.

Contents

1 Introduction

The fundamental task in tracking is to determine which observations at different time steps correspond to the same underlying object. The linkage or track initiation problem consists of making these determinations without any previous track estimates. Figure 1 illustrates the computational problem that we are trying to solve. Observations from five equally spaced 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 correspond to the desired motion model (Figure 1.B). The difficulty arises from the combinatorics of such a search.

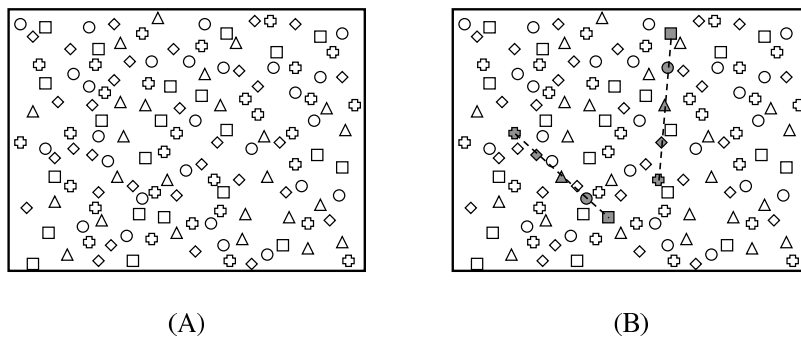(A)   (B)

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.

This is an important problem in such fields as target tracking and computer vision, but our primary motivating example in this paper is the asteroid linkage problem. Here we wish to determine which observed objects correspond to the same true underlying object from a series of visual observations of the night sky. These linkages can then be used to determine tentative orbits, attribute the observations to a known orbit, and assess the potential risk of an asteroid. The use of new observation techniques and equipment has increased the scope and accuracy of this problem, providing the potential to track hundreds of thousands of asteroids. The next generation of sky surveys, such as PanSTARRS or LSST, are designed to provide vast amounts of observational data that can be used to search for potentially hazardous asteroids. Further, these surveys have the potential to allow us to detect and track fainter objects. However, these improvements greatly increase the combinatorics of the problem reinforcing the need for tractable algorithms.

Below we introduce a new methodology for track initiation. Instead of treating track initiation as a sequential decision problem, we exhaustively consider all possible linkages. Thus we provide an exact algorithm for linkages. We then introduce a multiple tree algorithm for tractably finding the linkages. We compare this approach to an adapted version of multiple hypothesis tracking using spatial data structures and show

1

how the use of multiple trees can provide a significant benefit.

2 Problem Definition

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 initial estimates of the track parameters. Figure 2 shows a simple one dimensional example with five time steps and a linear model. The sets of linked observations are shown as open circles with their linear models as dashed lines.
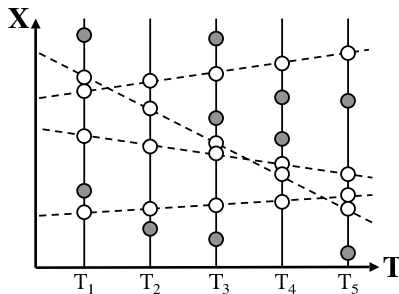

Figure 2: A set of one dimensional observations linked together by linear tracks. The white circles are the observations that correspond to the linear tracks (dashed lines).

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 observations consist of real-valued coordinates in D dimensional space, with $\mathbf{x}_i$ indicating the ith observation. These coordinates are the dependent variables of the track. We use t_i to indicate the independent variable of the ith observation. Although in many of the applications below t_i will correspond to the time of the observation, it can be used to represent any independent variable.

The second condition specifies a constraint on the observations' fit to the underlying model. A tuple of observations $(\mathbf{x}_{I_1}, \cdots, \mathbf{x}_{I_K})$ is valid only if there exists a track $\mathbf{g}$ such that:

$$\delta^L[d] \leq \mathbf{x}_{I_i}[d] - \mathbf{g}(t_{I_i})[d] \leq \delta^H[d] \quad \forall d, i \tag{1}$$

2

Equation 1 states that a track **g** is feasible for a tuple of observations if it falls within some bounds $[\mathbf{g}(t_{I_i})[d] + \delta^L[d], \mathbf{g}(t_{I_i})[d] + \delta^H[d]]$ of each observation $\mathbf{x}_{I_i}$ in each dimension d. The thresholds δ^L and δ^H provide upper lower bounds on the fit. Figure 3 shows an example of a feasible triplet using linear tracks and one feasible track for these points. The track is allowed to pass anywhere within the error bars around each point.
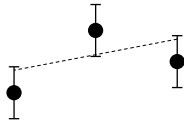
Figure 3: Three points that are compatible for linear tracks.

The above 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 4 shows an example of this. Further, we can vary δ^L and δ^H to account for systematic or time varying errors.
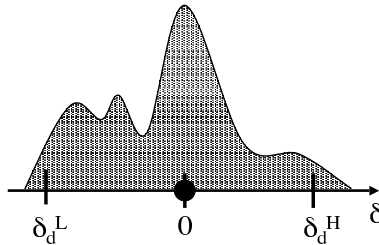
Figure 4: An arbitrary probability distribution and the resulting bounds. The circle denotes the observed location and the upper and lower bars indicate the acceptable locations for the track.

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. This is briefly discussed in Section 8.

Our discussion below focuses on two major types of tracks: linear and quadratic. The quadratic track is simply a quadric function of time:

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

3

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

$$\mathbf{g}(t) = \mathbf{b} \cdot t + \mathbf{c} \tag{3}$$

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.

3 Previous Work

There are a variety of different approaches to the problem of track initiation. Below we 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.

3.1 Sequential Track Initiation

One common approach to track initiation is sequential track initiation (for a good introduction see [Blackman and Popoli, 1999] and references therein). 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. There are many variations to this type of approach. One common and often successful variation is a very simple form of multiple hypothesis tracking. When a tentative track matches multiple observations at a given time step, multiple hypothesizes (tentative tracks) are formed and the decision 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 [Uhlmann, 1992, Uhlmann, 2001].

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. Early "good 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.

4
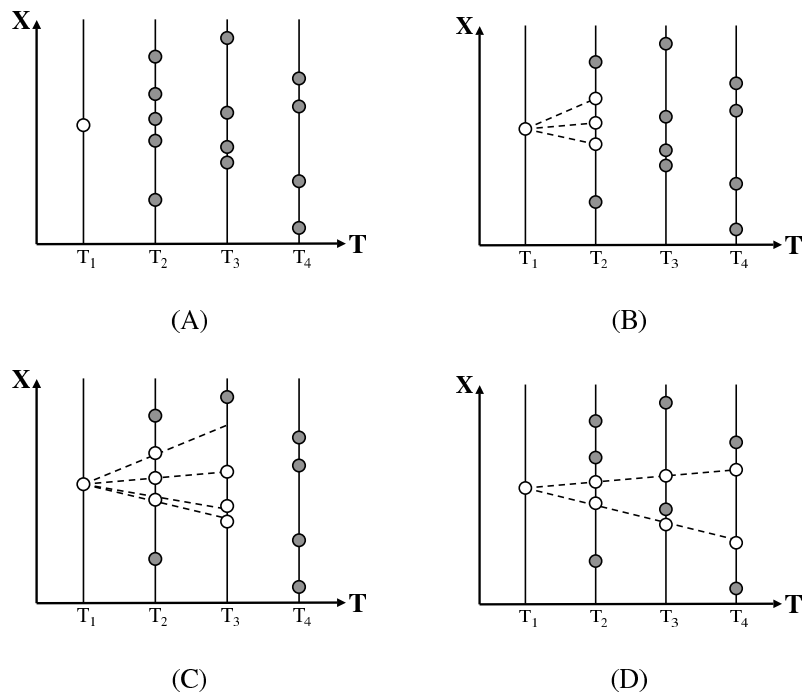
Figure 5: A multiple hypothesis tracker starts from a tentative track (A) and sequentially checks the later time steps. If multiple points fit a candidate track then several hypothesis are created (B) and (C).

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 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

5

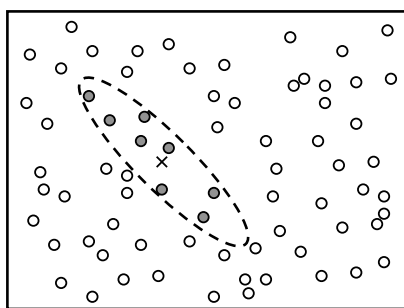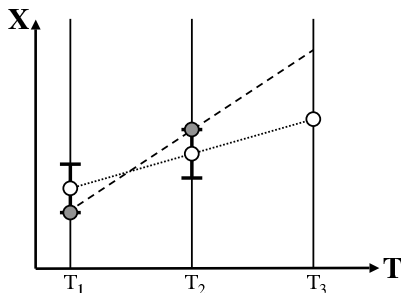Figure 7: Early noise in a track can significantly throw-off predicted positions. The true points are shown as open circles and the observed points are shown as shaded circles.

observations that are "obviously" members of other tracks.

3.2   Parameter Space Methods

Another approach to the problem of track initiation is to search for tracks in parameter space. One popular algorithm is the Hough transform [Hough, 1959]. The idea behind these approaches is that for many simple models, individual observations correspond to simple regions or curves in parameter space. An example with a linear model is shown in Figure 8. The points are shown in Figure 8.A and their corresponding lines in parameter space in Figure 8.B. If a series of observations lie along a line, then their lines in parameter space will intersect at a common point. The Hough transform looks for lines by using grid-based counts of the number of lines that go through a particular region of parameter space (Figure 8.C and 8.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. Secondly, 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 computational cost.

4   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
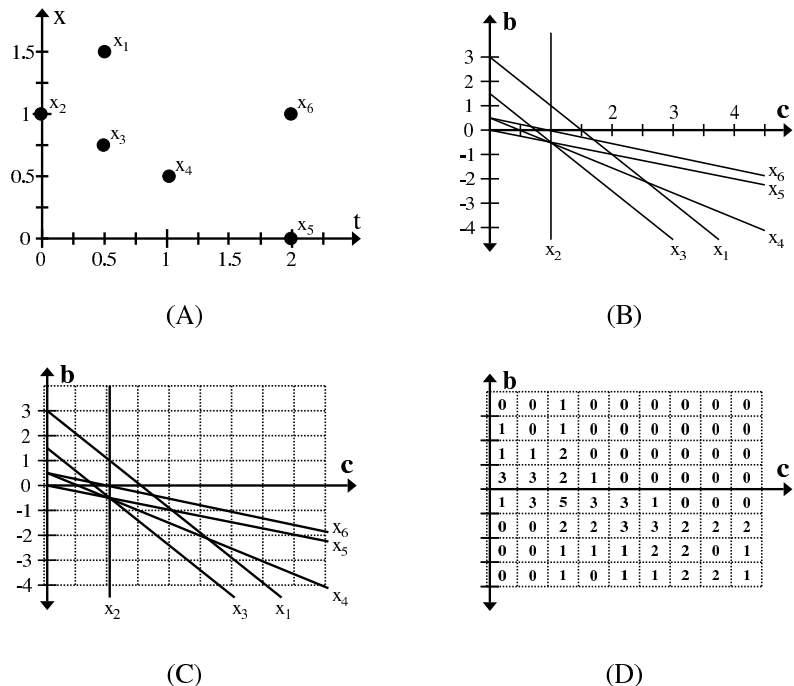
Figure 8: Under a linear model observations (A) correspond to lines in parameter space (B). If several observations lie along a line, their lines in parameter space will intersect at one place. Such clusters of intersections can be found by using grid based counts (C and D).

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 9 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 9.B). When the search reaches a point where all K trees are at leaf nodes, the algorithm explicitly tests the points at these nodes and returns those tuples of points that fit the criteria for being potential tracks (Figure 9.C). Figure 10 shows a snapshot of this traversal on two dimensional data. Figure 10.A and Figure 10.B show trees built from points at two different times. The dashed boxes indicate the current nodes being examined.

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 to fit a track through the K nodes. We call such sets of nodes *infeasible*. Such a

7

case is shown in Figure 9.D. This pruning criteria allows us to possibly ignore large numbers of the tuples that would have been tried under a brute force approach.
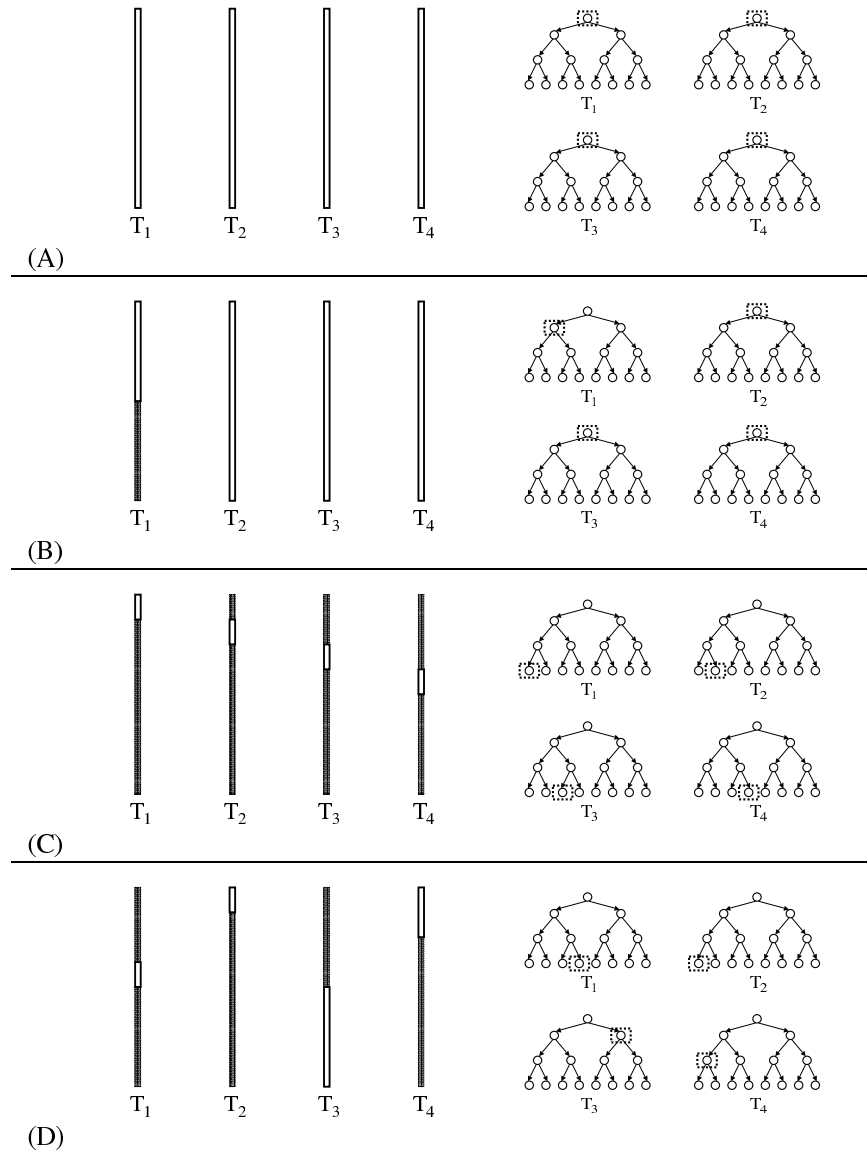

Figure 9: The multiple tree algorithm descends the trees in a depth first search (A and B). If it reaches the leaf nodes, it explicitly tests the tracks (C). The search can be pruned if it is not possible to fit a track through each of the nodes (D).

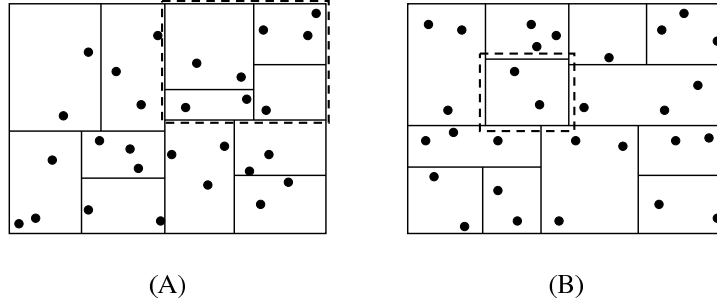There are multiple ways to choose which tree to descend. For now we use an

Figure 10: Two KD-trees built on two dimensional points at different times.

approach that chooses the tree with the most points below the current node. Other methods are discussed below.

It should be noted that the use of multiple trees has been explored for quickly answering spatial queries [Gray and Moore, 2001]. However, this work is 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 Pruning

In order for the multiple tree algorithm to be effective, we need to accurately and efficiently prune infeasible sets of nodes. Pruning is equivalent to asking: "Can there exist *any* track that passes through all K regions?" If no such track can exist, then we can safely stop searching these subtrees. In general, proof that such a track does or does not exist may be non-trivial to find.

Formally, the pruning question is equivalent to finding a set of track parameters $(\mathbf{a}, \mathbf{b}, \mathbf{c})$ that satisfy the constraints imposed by each node or proving that such parameters do not exist. We construct each KD-tree so that each node includes a bounding box that encloses the points that it owns. The bounds of the ith tree define $2D$ constraints on what qualifies as a feasible track, an upper bound $\mathbf{H}_i$ and lower bound $\mathbf{L}_i$ on the position at that time. In the quadratic case, these constraints are:

$$\mathbf{a}[1] \cdot t_i^2 + \mathbf{b}[1] \cdot t_i + \mathbf{c}[1] \leq \mathbf{H}_i[1]$$
$$\mathbf{a}[2] \cdot t_i^2 + \mathbf{b}[2] \cdot t_i + \mathbf{c}[2] \leq \mathbf{H}_i[2]$$
$$\vdots$$
$$\mathbf{a}[D] \cdot t_i^2 + \mathbf{b}[D] \cdot t_i + \mathbf{c}[D] \leq \mathbf{H}_i[D]$$
$$\mathbf{a}[1] \cdot t_i^2 + \mathbf{b}[1] \cdot t_i + \mathbf{c}[1] \geq \mathbf{L}_i[1] \tag{4}$$
$$\mathbf{a}[2] \cdot t_i^2 + \mathbf{b}[2] \cdot t_i + \mathbf{c}[2] \geq \mathbf{L}_i[2]$$
$$\vdots$$
$$\mathbf{a}[D] \cdot t_i^2 + \mathbf{b}[D] \cdot t_i + \mathbf{c}[D] \geq \mathbf{L}_i[D]$$

9

Thus we wish to find a vector $(\mathbf{a},\mathbf{b},\mathbf{c})$ that satisfies all $2KD$ such constraints.

We can greatly reduce the complexity of the problem by treating each dimension independently. This reduction is justified by the following theorem:

> **Theorem 1**: $(\mathbf{a},\mathbf{b},\mathbf{c})$ is a feasible track if and only if $(\mathbf{a}[i],\mathbf{b}[i],\mathbf{c}[i])$ satisfies the constraints in the ith dimension for all $1 \leq i \leq D$.
>
> **Proof**: The proof of Theorem 1 follows from the independence of the constraints. For each dimension, $1 \leq d \leq D$, we can create a set of constraints that only depend on the variables $\mathbf{a}[d]$, $\mathbf{b}[d]$, and $\mathbf{c}[d]$. These sets can be solved independently by choosing values for just those variables in the set. Since none of the other sets depend on those variables, the track $(\mathbf{a},\mathbf{b},\mathbf{c})$ is feasible if and only if $(\mathbf{a}[i],\mathbf{b}[i],\mathbf{c}[i])$ satisfies the constraints in the ith dimension for all $1 \leq i \leq D$.

Theorem 1 allows us to consider each dimension separately, reducing the pruning query with $2KD$ constraints to D sub-queries of $2K$ constraints. Further, the separation means that each sub-query consists of significantly fewer variables. For example, in the case of quadratic tracks each sub-query now consists of just 3 variables instead of $3D$.

Below we discuss a "smart brute force" search for answering the pruning queries. Although we restrict the discussion to the cases of linear and quadratic tracks, the results and discussion can be applied to tracks of other forms. The computational cost of pruning tracks of other forms may vary depending on the complexity of the track.

5.1 Brute Force Search

We use a "smart brute force" search to test for the existence of a feasible point. It should be noted that the constraints above can also be checked using linear programming. Despite this, the low number of variables and constraints and the existence of additional structure in the problem makes the smart brute force search computationally attractive.

Before describing the search algorithm, it is helpful to get intuition for the procedure by interpreting each constraint as a hyper-plane in parameter space. In the case of one dimensional quadratic tracks, the constraint forms a plane in 3-dimensional space (a,b,c) and for linear tracks the constraint forms a line in 2-dimensional space (b,c). Each constraint thus defines a partitioning of parameter space into a half-space of feasible points, which lie on one side of the hyper-plane, and a half-space of infeasible points, which lie on the other. If the intersection of the feasible half-spaces is not empty, then there exists a track that satisfies all of the constraints. An example with linear tracks and 6 constraints is shown in Figure 11. The tracks that satisfy all of the constraints occupy the unshaded region of parameter space.

In its simplest form, our search consists of checking the "corners" of the constraints for a feasible point. In a C-dimensional parameter space, two C-dimensional hyper-planes intersect at a $(C-1)$-dimensional hyper-plane and C non-parallel hyper-planes will intersect at a point. We call this point a corner. Since the hyper-planes define the boundary of the feasible region, this region is non-null if and only if one such corner exists and is feasible:
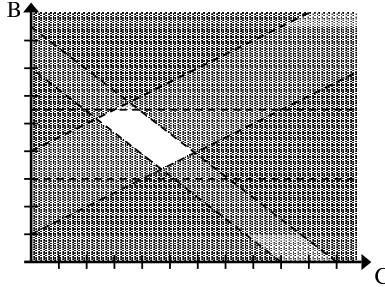
Figure 11: Tracks that conform to all of the constraints lie in the region of parameter space that is defined by the intersection of the constraint half-spaces. The feasible half-spaces are the unshaded regions.

Theorem 2: The intersection of K half-spaces defined by at least C non-parallel C-dimensional hyper-planes is not empty if and only if there exists a point $\mathbf{x}$ such that $\mathbf{x}$ is feasible and lies on at least C hyper-planes.

Proof: It is easy to see that if there exists a feasible track $\mathbf{x}$ then the intersection of the K half-spaces is not empty regardless of where $\mathbf{x}$ lies. Thus we only need to show that if the intersection is not empty then there exists a feasible track $\mathbf{x}$ such that $\mathbf{x}$ lies on at least C hyper-planes.

Let $\mathbf{y}$ be an arbitrary feasible track, which by definition lies in the intersection of K half-spaces. Assume that $\mathbf{y}$ lies on $c < C$ nonparallel boundary hyper-planes. Since the c hyper-planes intersect at a $C' = C - c$ dimensional hyper-plane, $\mathbf{y}$ can be any feasible point on that C'-dimensional hyper-plane. Therefore we can move $\mathbf{y}$ to be a new point $\mathbf{y}'$ by sliding $\mathbf{y}$ along this hyper-plane until it intersects a new constraint. Such an intersection must exist because there are at least C nonparallel hyper-planes. Further, if we constrain $\mathbf{y}$ to move to the intersecting hyper-plane that is closest (requires the least translation along the C'-dimensional hyper-plane), then we do not cross any other hyper-planes and $\mathbf{y}'$ is still a feasible point. Thus $\mathbf{y}'$ is a feasible point that lies on $c + 1$ boundary hyper-planes. We can continue to push the feasible point in this manner until it lies on C nonparallel hyper-planes. Thus if the intersection is not empty then there exists a feasible track $\mathbf{x}$ such that $\mathbf{x}$ lies on at least C hyper-planes.

It should be noted that Theorem 2 does not require that the feasible region is fully enclosed by the hyper-planes. Instead, it only requires that there exists at least C nonparallel hyper-planes. Under this condition there will be at least one corner to search and the theorem holds. Figure 12 shows such a set. If there is only $C' < C$ nonparallel hyper-planes then by the same reasoning the feasible region is non-empty if and only if *any* point on the intersection of those C' nonparallel hyper-planes is feasible.

We can use Theorem 2 to define a brute force search for a feasible point. Specif-
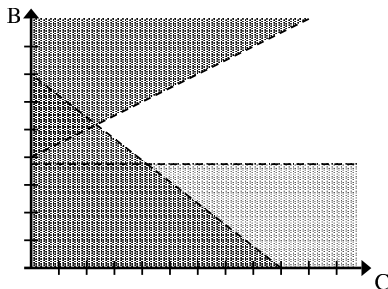
11

Figure 12: The search algorithm will work as long as there are at least C nonparallel planes, even if the feasible region is unbounded.

ically, we can test each C-tuple of nonparallel hyper-planes and calculate the point of intersection. We can then test whether this point is feasible, by testing it against each of the constraints. This search requires $O(K^{(C+1)})$ operations, $O(K)$ feasibility tests for each of the $O(K^C)$ corners. Below we discuss ways that we can exploit the structure of the problem to reduce this cost.

Above we make the implicit assumption that the planes are not all parallel or in another degenerate configuration. In general these additional cases can easily be handled as special cases. It is important to note though that many of the planes will have at least one corresponding parallel plane. Specifically, the upper and lower constraints for any node in any dimension form two parallel planes:

$$\mathbf{a}[d] \cdot t_i^2 + \mathbf{b}[d] \cdot t_i + \mathbf{c}[d] \leq \mathbf{H}_i[d]$$
$$\mathbf{a}[d] \cdot t_i^2 + \mathbf{b}[d] \cdot t_i + \mathbf{c}[d] \geq \mathbf{L}_i[d] \tag{5}$$

5.2 Utilizing Structure in the Search

In general the above brute force search is too computationally expensive to be used for the pruning query. We can mitigate this cost by noting that the tree search provides a significant amount of structure that can be exploited:

1. At each level of the search, the constraints for all tree nodes except one are identical to the previous level.

2. At each level of the search, the constraints for the one tree node that changed are *tighter* than at the previous level.

The first observation follows from the fact that we are only splitting one node at each level of the search. Thus all of the other nodes and their bounds remain unchanged. The second observation follows from the fact that each time we split a node, the position bounds of the children nodes are strictly contained within the position bounds of the parent node. Therefore the constraints defined by these bounds always get tighter.

The first observation indicates that we can avoid the computation altogether if the feasible track found at the previous level is compatible with the few new constraints. We must have already found a feasible track for the last level by virtue of the fact we did not prune at the previous level of the search. Thus we can keep this point and test it against the new constraints. If it remains compatible, we do not need to resolve all of the constraints.

The first and second observation also indicate that we only need to search a limited number of corners if the old point is not feasible. Specifically, if the old point is not compatible with a new constraint then either the new set of constraints are not compatible *or* a new feasible point lies on a corner where one of the planes is the incompatible constraint. This is shown below in Theorem 3.

> **Theorem 3**: If the feasible track from the previous level is not compatible with a new constraint then either the new set of constraints is not compatible *or* a new feasible point lies on the plane defined by the new constraint.
>
> **Proof**: We prove this by showing that if the set of constraints is compatible and the feasible track from the previous level $\mathbf{x}$ is not compatible with the new constraint then there exists a new feasible point $\mathbf{x}'$ that lies on the plane defined by the new constraint. Let $\mathbf{y}$ be any point in the new feasible region. Since the new constraint is strictly tighter $\mathbf{y}$ also lies in the previous feasible region. We can define a line segment L between $\mathbf{x}$ and $\mathbf{y}$. Since the feasible regions are convex, all points on L lie within the feasible region from the previous level. Further, because $\mathbf{x}$ is not in the new feasible region, the line segment must intersect the hyper-plane defined by the new constraint. This point of intersection is $\mathbf{x}'$.

Theorem 3 indicates that we do not need to search all corners, but rather only the corners that contain the infeasible constraint. Further, we can add new constraints one at a time and only do the search if the previous point is not feasible. This reduces the cost of handling an infeasible point from $O(K^{C+1})$ to $O(K^C)$.

5.3 Combining Operations

We can further reduce the cost of pruning by combining the searching and checking steps. Here we can take advantage of the fact that with a C-dimensional parameter space, $C-1$ nonparallel hyper-planes intersect at a line. Instead of searching all corners we can instead search all lines. Then in order to check feasibility (and find a specific feasible point) we can test the other constraints against that line. Each constraint will either be parallel to the line, and can be checked directly, or will intersect the line. Each intersection provides a signed point on the line. We can examine all of these signed intersections asking whether *any* point on the line is feasible. An example is shown in Figure 13.

This joint searching and merging steps means that we only need to check $O(K^{C-1})$ lines instead of $O(K^C)$ corners. Further, it can be combined with the tricks in Section 5.2 to reduce the cost of handling an *infeasible* point from $O(K^{C+1})$ to $O(K^{C-1})$. While this savings may not appear significant, for the relatively simple task of quadratic
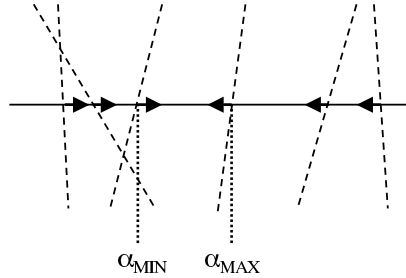
$$\alpha_{MIN} \qquad \alpha_{MAX}$$

Figure 13: We can check for feasibility along a line by checking the signed intersections of each constraint and the line.

tracks ($C = 3$) and five time steps ($K = 5$) these improvements provide a factor of 25 speedup.

5.4 Additional Constraints

The above pruning methodology provides a simple and formal way to provide additional constraints for the tracks. For example, we may wish to provide bounds on the minimum and maximum accelerations that a target can undergo. These additional constraints can be specified directly:

$$\mathbf{a}[d] \leq \mathbf{a}^{MAX}[d]$$
$$\mathbf{a}[d] \geq \mathbf{a}^{MIN}[d] \tag{6}$$

and fit into the pruning algorithm without modification. This allows the user to seamlessly provide potentially valuable domain knowledge.

5.5 Missing Observations

Up to this point, our discussion has assumed that each track produces one observation *every* time step. There are several simple approaches to handling missing observations. Perhaps the easiest approach is include "missing" as a single new node in the tree as shown in Figure 14. Additional logic can be added to prune the search if too many trees are at the "missing" node, preventing such problems as returning tracks without sufficient support. Here care must be taken to avoid adding subsets of valid tracks that have already been found. Finally, allowing too many missing points may greatly increase the computational load by performing the search repeatedly.

6 Relation to Conventional Track Initiation

The multiple tree algorithm can be adapted to function in a manner similar to sequential track initiation, albeit with a different search order than is normally used. Consider the
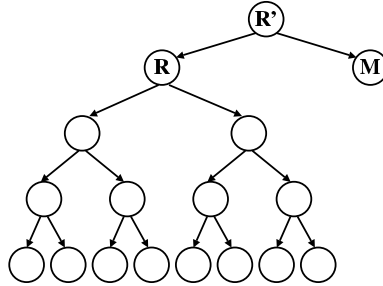
14

Figure 14: One approach to handling missing points is to treat "missing" as another tree node. Here the new tree is shown is a new root R' and a missing node M.

following rule for descending the *K* trees:

> Always descend the earliest tree that is not already at a leaf node.

This rule 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. Thus the decent rule makes the multiple tree algorithm perform like the sequential track initiation algorithms.

It is important to note that although this algorithm performs similarly to the sequential track initiation, it is still an exact algorithm for detecting potential tracks. Unlike many proposed sequential track initiation algorithms, it does not try to fit a track to the first few points and project this track ahead in time. Instead it maintains the same feasibility criteria as the general multiple tree algorithm and will always return the same result. For this reason, we use this descent rule for comparison in the below experiments.

7 Experiments

7.1 Algorithms

In the below experiments, we examine a series of variations on the multiple tree algorithm, denoted MT-1 through MT-K. Specifically, we use a rule similar to the one described in Section 6:

> **Descent Rule for MT-k**: If one or more of the first *k* trees are not at a leaf node, 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 node.

15

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 trees. When $k = 1$, this algorithm is the one described in Section 6 and mimics sequential track initiation. These approaches cover the spectrum from a conventional track initiation approach to a full K-tree algorithm.

7.2 Simulated Data

For the first set of experiments we used observations generated from artificial tracks in order to examine the algorithms' relative performance under a variety of conditions. The data was generated by first creating N artificial quadratic tracks:

- $\mathbf{c} \sim \text{uniform}(0, 1)$

- $\mathbf{b} \sim \text{uniform}(-\mathbf{b}_{MAX}, \mathbf{b}_{MAX})$

- $\mathbf{a} \sim \text{uniform}(-\mathbf{a}_{MAX}, \mathbf{a}_{MAX})$

The bounds on velocity and acceleration were included as constraints on feasible tracks. Observations were then generated by sampling the tracks at each time step t:

$$\mathbf{x}_i^t[d] = \frac{1}{2}\mathbf{a}_i[d]t^2 + \mathbf{b}_i[d]t + \mathbf{c}_i[d] + \varepsilon \; \forall d : 1 \leq d \leq D \tag{7}$$

where ε is drawn uniformly from $[\delta^L[d], \delta^H[d]]$ for all d. In the below experiments $\delta^L[d] = -0.01$ and $\delta^H[d] = 0.01$.

Using the simulated observations, we can then ask about the benefit of using multiple trees as we vary different parameters in the data set. Since the algorithms are all exact algorithms, they will return the same set of solutions. The major difference then is the number of pruning computations that are performed throughout the run.

7.2.1 Number of Tracks

The primary factor that we would expect to influence the performance of the algorithms is the number of tracks. We varied the number of tracks from 50 to 5000 and compared the number of pruning queries from each of the algorithms. The average results over 30 trials are shown in Table 1. As shown, MT-3 consistently outperforms the other algorithms.

Unfortunately, the performance benefit is largely offset by the increased density of points. The high density of points means that there are less "empty" regions of space and thus less pruning is done because more of the initial tracks appear feasible. Table 2 shows the same experiment as Table 1, but with "slower" tracks. The decrease in maximum velocity and acceleration effectively reduces the density of the points relative to their motion. As shown this change significantly aids all algorithms, but especially helps the multiple tree algorithms.

16

N	MT-1	MT-2	MT-3	MT-4
50	2.86×10^3	2.54×10^3	2.28×10^3	2.59×10^3
100	7.60×10^3	6.85×10^3	6.21×10^3	8.37×10^3
500	1.11×10^5	1.05×10^5	8.41×10^4	1.52×10^5
1000	4.66×10^5	4.50×10^5	3.30×10^5	5.81×10^5
2000	2.37×10^6	2.32×10^6	1.72×10^6	2.60×10^6
5000	2.57×10^7	2.56×10^7	2.08×10^7	2.24×10^7

Table 1: The average number of pruning tests for simulated quadratic tracks as the number of tracks increases. These tests use "fast" tracks: $\left|\mathbf{b}[d]\right| \leq 0.1$ and $\left|\mathbf{a}[d]\right| \leq 0.05$.

N	MT-1	MT-2	MT-3	MT-4	MT-5
50	2.46×10^3	2.09×10^3	1.73×10^3	1.39×10^3	1.07×10^3
100	5.74×10^3	4.83×10^3	4.02×10^3	3.30×10^3	2.60×10^3
500	3.88×10^4	3.22×10^4	2.62×10^4	2.21×10^4	2.01×10^4
1000	8.73×10^4	7.24×10^4	5.95×10^4	5.35×10^4	5.56×10^4
2000	1.98×10^5	1.65×10^5	1.36×10^5	1.33×10^5	1.60×10^5
5000	6.11×10^5	5.17×10^5	4.25×10^5	4.87×10^5	7.28×10^5

Table 2: The average number of pruning tests for simulated quadratic tracks as the number of tracks increases. These tests use "slow" tracks: $\left|\mathbf{b}[d]\right| \leq 0.01$ and $\left|\mathbf{a}[d]\right| \leq 0.005$

7.2.2 Gap Between Observations

Without an initial estimate of velocity or acceleration, the time between observations may significantly affect performance. If either the movements or the temporal gaps are small, then the next point on the track will often be close to the location of the last point even without accounting for the movement. As the gaps or velocities increase we may have to try *many* neighbors at the next time step to find the true neighbor. We would expect to see an increased benefit from using multiple trees as the velocity or time between observations increases. To test this, we generated artificial linear tracks with a fixed range of velocities ($-0.1 \leq \mathbf{b}[d] \leq 0.1$) and increased the spacing in time of the observations. The results are shown in Table 3.

Table 3 shows one of the primary benefits of the multiple tree approach. As the gap in time increases, the number of pairs of observations in the first two time steps that comply with the velocity constraints increases. The use of three trees prevents us from having to examine many of these pairs by incorporating information from later time steps. However, after three trees the track is relatively well confirmed and additional trees do not help.

Δt	MT-1	MT-2	MT-3	MT-4	MT-5
0.01	5.16×10^5	4.16×10^5	3.20×10^5	2.32×10^5	1.51×10^5
0.05	5.79×10^5	4.86×10^5	3.88×10^5	3.86×10^5	4.38×10^5
0.10	7.61×10^5	6.69×10^5	4.86×10^5	6.00×10^5	8.56×10^5
0.20	1.44×10^6	1.34×10^6	7.22×10^5	1.15×10^6	2.09×10^6
0.50	5.30×10^6	5.75×10^6	1.97×10^6	3.95×10^6	8.18×10^6
1.00	2.08×10^7	2.06×10^7	5.52×10^6	1.13×10^7	2.27×10^7

Table 3: Average results for the simulated data sets with linear tracks ($\mathbf{b}_{MAX}[d] = 0.1 \ \forall d \geq 1$) with 5000 observations at 5 times with varied temporal spacing.

K	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 4: The number of pruning tests (in millions) for the astronomy data with varying numbers of observation and time step size.

7.3 Astronomy Data

In addition to completely artificial data, we examined simulated data from the astronomy domain in order to test whether this algorithm provides a benefit on tracks of the distribution of real asteroids. Specifically, we 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 consisted of two components, Right Ascension and declination, that gave the object's location in the sky. We used this approach to generate observations from 4, 5, and 6 time steps equally spaced over the 16 nights and covering a 1 square degree region of the sky. This region included observations for 1,768 different objects.

Table 4 show the results of this experiment. The differences between performance of different K is due to both a varying number of observations and the different spacing between observations. As shown, the use of multiple trees can lead to a significant reduction in the number of pruning queries needed.

As discussed in Section 7.2 and indicated by the previous experiment, the spacing in observation times can have a significant effect on the number of neighbors we need to search at the next time step. To examine this trend on the astronomy data, we examined how many observations fell between the projected position and the true position using the simulated astronomy data. Observations were generated from the quadratic tracks at varying time intervals. Each observation was then projected to the next time step (without velocity) and the distance to the location of the true next observation was calculated as δ. Finally, the number of points closer to the predicted position was

18

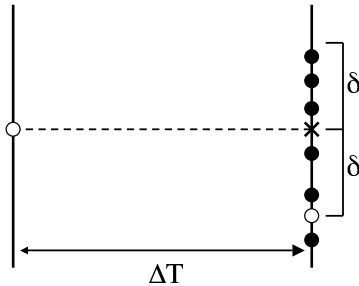calculated. Figure 15 shows an illustration of the test.

Figure 15: 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.

Figure 16 shows the results of varying the time between observations. For each spacing, the graph indicates a histogram of the position of the true closest neighbor. At a spacing of 1 day most of the observations are very close to the true next point. As the gap in time increases the distance to the closest point rapidly increases and more neighbors at the next time step would have to be searched with a sequential algorithm. With a spacing of 4 days, many of the tracks would require searching many observations to find the true completion of the track. Further, the right hand side of the graph shows a sharp increase in the number of tracks whose neighbor is *further* than 100 points away. It should also be noted that since we are dealing with only a subset of the sky, these histogram provide an *optimistic* estimate. In real data the points lying along the edge of the region will also have interference coming from outside the region.

The trend shown in Figure 16 confirms one of the primary advantages of using multiple trees. As the spacing in time increases, many neighbors may have to be searched in the early time steps. The use of multiple trees mitigates this problem by combining pruning information for later time steps, thus contributing to the speedups shown in Table 4.

8   Unknown or Complex Track Models

The above discussion assumes that we have a known and relatively simple track model. However in many domains, this may not the be case. In the astronomy domain, the true tracks of the asteroids across the sky are not quadratic. For example, relative motion of the earth may cause the track to undergo retrograde motion.

The easiest solution to the problem of a poor or unknown track model is to approximate the track with a simple model. This often requires a relatively short time span. Fortunately, this complements the track initiation query itself where we are interested in finding a set of observations to indicate the start of a track. If longer time spans are needed, then it is possible to use the above algorithms to find short arcs that can then be
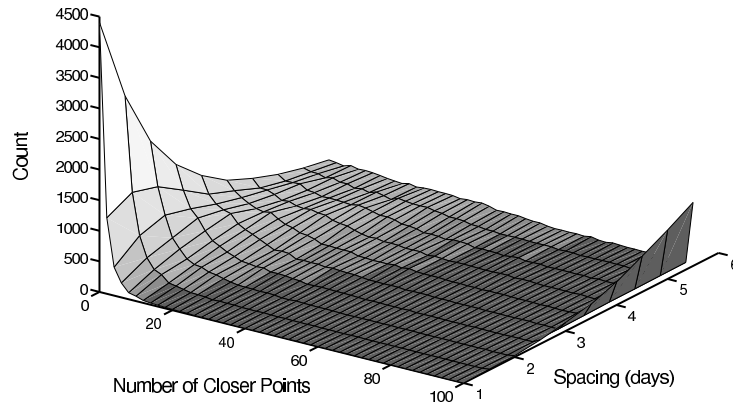
Figure 16: Histogram of true neighbor positions for each gap in observation time.

glued together by a conventional tracking algorithm. Such an approach was suggested by Shaw and Arnold [Shaw and Arnold, 1995]. They proposed gluing together track segments found by a dynamic programming track-before-detect method. Finally, we can account for systematic errors in the model by extending the fit threshold. However, it should be noted that if the fit threshold is too loose then many false positives will be returned.

9 Conclusions

Above we describe an exhaustive methodology for track initiation. We introduced a multiple tree algorithm for tractably finding the linkages. Empirically, this algorithm performed very well on several simulated data sets, outperforming an exact adaptation of conventional multiple hypothesis tracking.

The true advantage of the multiple tree algorithm lies in its ability to use information from later time steps to aid in pruning decisions at earlier time steps. For example, an exhaustive method that does not account for this information may try *every* pair of observations from the first two time steps when using a quadratic model. The additional pruning information afforded by the use of multiple trees can become even more significant as the time between observations increase and bounds on track parameters (such as maximum velocity) become weaker. In addition, the use of multiple trees provides the ability to reduce redundant computation that can arise from pruning for similar points or initial tracks.

20

Acknowledgements

References

[Blackman and Popoli, 1999] Samuel Blackman and Robert Popoli. *Design and Analysis of Modern Tracking Systems*. Artech House, 1999.

[Gray and Moore, 2001] Alexander Gray and Andrew Moore. N-body problems in statistical learning. In Todd K. Leen and Thomas G. Dietterich, editors, *Advances in Neural Information Processing Systems*. MIT Press, 2001.

[Hough, 1959] P. V. C. Hough. Machine analysis of bubble chamber pictures. In *International Conference on High Energy Accelerators and Instrumentation*. CERN, 1959.

[Shaw and Arnold, 1995] Scott W. Shaw and James F. Arnold. Design and implementation of a fully automated oth radar tracking system. In *Proceedings of the IEEE International Radar Conference*, pages 294–298, May 1995.

[Uhlmann, 1992] J. K. Uhlmann. Algorithms for multiple-target tracking. *American Scientist*, 80(2):128–141, 1992.

[Uhlmann, 2001] J. K. Uhlmann. Introduction to the algorithmics of data association in multiple-target tracking. In David L. Hall and James Llinas, editors, *Handbook of Multisensor Data Fusion*, pages 3.1–3.18. CRC Press, 2001.