

A Multiple Tree Algorithm for the Efficient Association of Asteroid Observations

Jeremy Kubica
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA USA
jkubica@ri.cmu.edu

Andrew Connolly
University of Pittsburgh
Pittsburgh, PA USA
ajc@phyast.pitt.edu

Andrew Moore
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA USA
awm@cs.cmu.edu

Robert Jedicke
University of Hawaii
2680 Woodlawn Drive
Honolulu, HI 96822
jedicke@ifa.hawaii.edu

ABSTRACT

In this paper we examine the problem of efficiently finding sets of observations that conform to a given underlying motion model. While this problem is often phrased as a tracking problem, where it is called track initiation, it is useful in a variety of tasks where we want to find correspondences or patterns in spatial-temporal data. Unfortunately, this problem often suffers from a combinatorial explosion in the number of potential sets that must be evaluated. We consider the problem with respect to large-scale asteroid observation data, where the goal is to find associations among the observations that correspond to the same underlying asteroid. In this domain, it is vital that we can efficiently extract the underlying associations.

We introduce a new methodology for track initiation that exhaustively considers all possible linkages. We then introduce an exact tree-based algorithm for tractably finding all compatible sets of points. Further, we extend this approach to use multiple trees, exploiting structure from several time steps at once. We compare this approach to a standard sequential approach and show how the use of multiple trees can provide a significant benefit.

Categories and Subject Descriptors

E.1 [Data Structures]: Trees; J.2 [Physical Sciences and Engineering]: Astronomy

General Terms

Algorithms

Keywords

Multiple Tree Algorithms, Track Initiation

1. INTRODUCTION

A common task in both tracking and data mining is to find sets of observations or data points that fit some underlying model. In the domain of tracking, the track initiation problem consists of using this approach to determine which observations from different time steps correspond to the same underlying object 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 marked with different numbers. 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.

While this problem is important to such tasks as target tracking, computer vision, and mining spatial-temporal data sets, our primary motivating example in this paper is asteroid linkage. Here we wish to determine which observations correspond to the same true underlying asteroid. 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 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 than ever before. However, these improvements greatly increase the combinatorics of the problem reinforcing the need for tractable algorithms.

Current approaches for finding asteroid linkages are not suitable for these new surveys. The current tools are computationally impracticable at the magnitude of the new problem. Further, previous approaches were designed to work on relatively clean data, such as bright objects observed at closely spaced intervals. In contrast, we present an approach designed to work on many objects that are observed at potentially widely spaced times.

Below we introduce a new methodology for asteroid linkage and track initiation. We present an approach that exhaustively considers *all* possible linkages and returns *all* linkages that conform to our constraints. We then introduce a tree-based algorithm for tractably finding the linkages and provide an efficient pruning methodology that exploits the structure of both the problem and the search. We further extend this approach to use multiple trees, exploiting structure from several time steps at once. We compare our multiple tree

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

KDD'05, August 21–24, 2005, Chicago, Illinois, USA.
Copyright 2005 ACM 1-59593-135-X/05/0008 ...\$5.00.

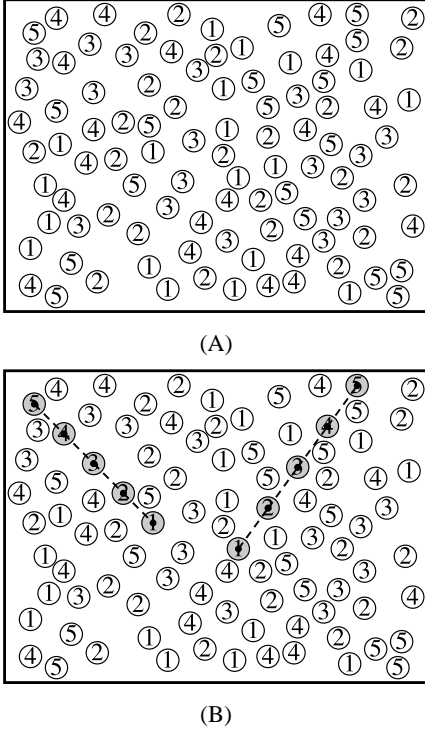


Figure 1: The linkage problem consists of finding one point from each time step such that they fit the given model (e.g. in (A) we must find five points numbered 1-5 that are equally spaced along a line). Two linear tracks are shown (B) and a third is left as an exercise for the reader.

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.

2. PROBLEM DEFINITION

The linkage problem consists of finding *all* tuples of points that conform to an underlying motion model. Thus the problem is similar to a *spatial join* query, extended to find associations relative to a given set of motion models. Formally, we can specify the problem as a search query. At each time step m we observe N_m points from both the underlying set of objects and a noise process. Given a set of observations at M distinct time steps, we want to find and return all tuples of observations such that:

1. there is 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.

The observations consist of real-valued coordinates in D dimensional space, with \mathbf{x}_i indicating the i th observation, and time of observation t_i . Although in many of the applications below t_i will correspond to time, it can represent any independent variable.

The second condition requires that the observations fit within some bounds $[\delta^L, \delta^H]$ of the underlying model. Formally, a tuple of observations $(\mathbf{x}_{t_1}, \dots, \mathbf{x}_{t_M})$ is valid only if there exists a track \mathbf{g} such that:

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

This condition is not tied to a particular statistical model. We can define an arbitrary distortion model for the observations and set the thresholds as the 99.9% confidence interval for the noise in this dimension. Further, we can vary δ^L and δ^H to account for systematic or time varying errors.

For simplicity and consistency our discussion below focuses on two types of models: *quadratic* and *linear*. These models naturally describe the motions of objects with constant acceleration and constant velocity respectively. Both tracks can be represented as:

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

with the additional restriction of $\mathbf{a} = \mathbf{0}$ for linear tracks.

3. PREVIOUS WORK

Below we briefly discuss a few of the more common approaches to track initiation and asteroid linkage. These approaches differ from our own in several important ways. We are asking for *all* sets of observations that could feasibly belong to a path and we provide an exact algorithm for answering this query.

3.1 Sequential Algorithms

One common approach to track initiation is a sequential approach (for a good introduction see [1, 3]). Unassociated points are treated as the start of a new track and projected to later time steps where they are associated with other points to form longer tracks. It is possible to use this approach to find all feasible linkages by incorporating a very simple form of multiple hypothesis tracking [7]. Specifically, whenever a track could feasibly be associated with multiple different observations at a given time step, we allow the search to branch off and try each new tentative track (one for each association). This process is illustrated in Figure 2. The initial point matches three other points at the second time step, which are used to create three hypothesized tracks. This process continues to later time steps with “bad” hypotheses being pruned away.

In order to reduce the number of candidate neighbors examined at each time step, *gating* is used to eliminate the obviously infeasible associations. As shown in Figure 3, new points are first filtered by whether they fall within a window or gate around the model’s predicted position. This approach has also been used in conjunction with kd-tree structures [2] to quickly retrieve the candidate observations near the predicted position of a track [11, 12].

It should be noted that 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 process. While branching and considering multiple associations mitigates this problem, it introduces another problem, the possibility of a high branching factor causing a significant computational load.

Below we extend this multiple hypothesis tracking approach to our exhaustive search criteria. Instead of relying on an estimated track and predicted position, our search tests points at the new time step for compatibility with the current track (set of previous observations) by explicitly determining whether they satisfy Equation 1. To this end we propose new pruning criteria for a tree-based search that allows us consider this type of satisfiability constraint.

3.2 Parameter Space Methods

Another approach to this problem is to search for associations in the parameter space of the models themselves. One popular algorithm is the Hough transform [6]. The idea behind these approaches is that for many simple models, individual observations correspond

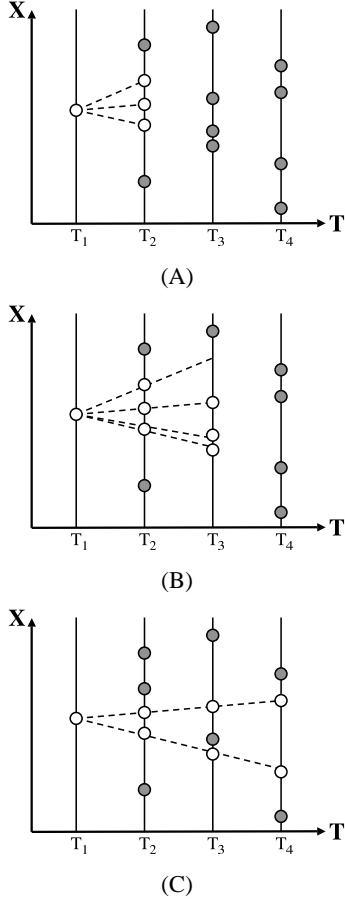


Figure 2: A multiple hypothesis tracker starts from a tentative set of associations and sequentially checks the later time steps (A). If multiple points reasonably fit the model then several hypotheses are created (B) and (C).

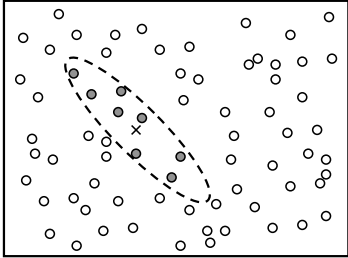


Figure 3: Gating can be used to ignore points that could not be part of the current track. The X marks the model's predicted position and the points that fall within the gate are shaded.

to simple regions or curves in parameter space. For example when linear models are used, a single point in observation space (t, x) can be transformed into a line in parameter space:

$$c = (-t) \cdot b + x \quad (3)$$

If a series of observations actually lie along a line, then their corresponding lines in parameter space will intersect at a common point (namely, the point representing the parameters of the line on which

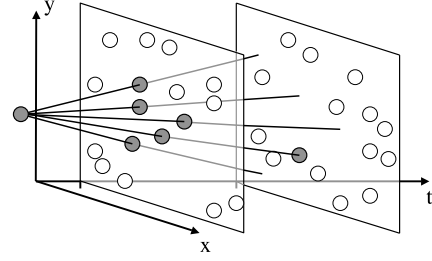


Figure 4: One potential advantage of using multiple trees is that we can use information from later time steps to prune out bad associations at earlier time steps. There are five initial pairings that appear reasonable for a given query point, but only one is confirmed with a later observation.

they lie). The Hough transform looks for such intersections in parameter space by using grid-based counts of the number of lines that go through a particular region of parameter space.

There are several major downsides to the parameter space approach. 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 may require significant amounts of space.

3.3 Tracking Dense Point Sets

A significant amount of work exists in the computer vision and tracking community for the related question of tracking a dense set of point observations. Here the goal is to find a set of “good” associations for each observation by assigning the observations to tracks and scoring the set of tracks with such metrics as smoothness or consistency. Many of the proposed techniques use a sequential approach, scanning through time making or refining track/observation assignments. For example, the greedy exchange algorithm [9] and its enhancements (e.g. [8, 13]), start with an initial set of assignments and iteratively scan through the time steps looking for pairs of observations from different tracks that can be exchanged. This algorithm continues to make such adjustments until they no longer improve the assignments.

The algorithms developed in these fields may provide accurate and efficient ways to track dense sets of observations. However, the important difference between this work and our own is in the actual question that we are trying to answer. The approaches described above attempt to return observation/track assignments that produce a “good” set of disjoint tracks. Therefore one limiting assumption that appears in these algorithms is that each observation can only be assigned to at most one track. In contrast, we are interested in returning *all* feasible sets of observations. Thus a more comprehensive search is required.

4. MULTIPLE TREE ALGORITHM

Our solution to this problem 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. This potential advantage is illustrated in Figure 4.

Figure 5 shows a one dimensional example of this approach. One tree is built independently on each of the M time-steps. The algo-

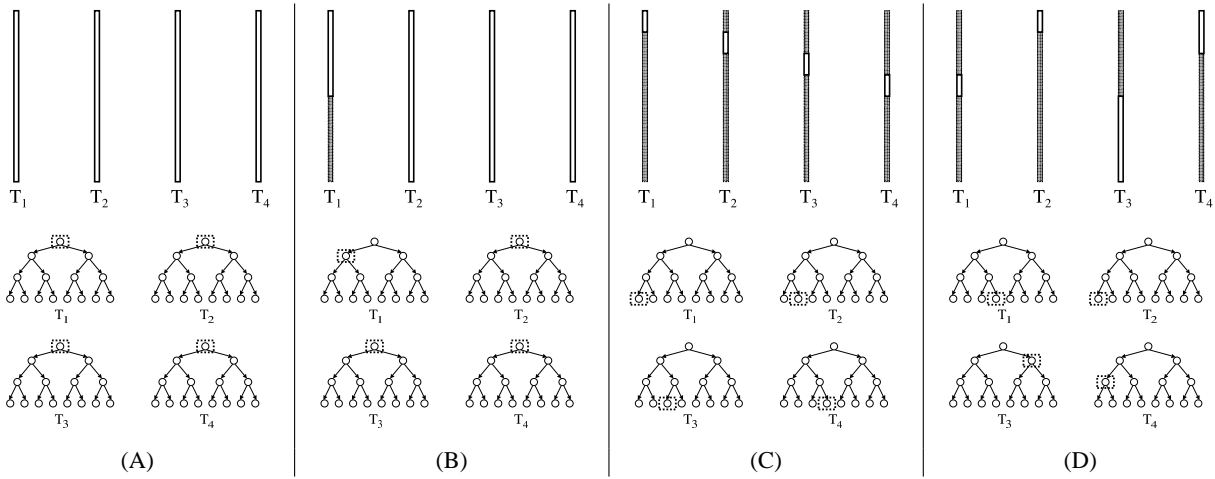


Figure 5: The multiple tree algorithm descends the trees in a depth first search (A and B). If it reaches the leaf nodes, it explicitly tests all sets of observations (C). The search can be pruned if it is not possible to fit our model through each of the nodes (D).

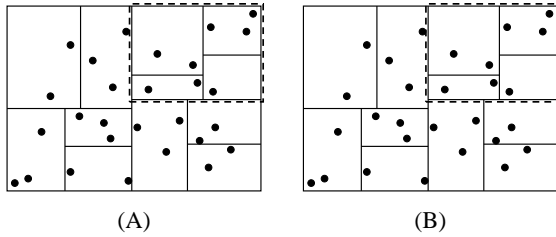


Figure 6: A snapshot of the multiple tree traversal of two KD-trees built on two dimensional points at different times. The dashed boxes indicate the current nodes being examined.

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 5.B). When the search reaches a point where all M 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 of our model (Figure 5.C). Figure 6 shows a snapshot of this traversal on two dimensional data.

In the above form, the algorithm would search all combinations of the $\sum_{m=1}^M N_m$ leaf nodes, requiring $O(\prod_{m=1}^M N_m)$ time. The benefit of using the tree-based algorithm is that we can prune regions of the search space if we can ever show that *no* set of points from the current nodes could be compatible with our model. We call such sets of nodes *infeasible*. Such a case is shown in Figure 5.D. This pruning criteria allows us to possibly ignore large numbers of the tuples that would have been tried under a brute force approach.

The full recursive algorithm is given in Figure 7. It is initially called using the set of root nodes from the trees at each time step. We describe the pruning test in detail in Section 5. It should also be noted that the decision of which tree to descend (line 5) can be replaced with other criteria. Of particular interest is the condition that we always descend the *first* non-leaf tree node. This alternate criterion transforms the search to consider the time steps sequentially. Thus it is an exhaustive version of a sequential tracking algorithm adapted to use our feasibility criteria and tree-based pruning.

Multiple Tree Track Initiation

Input: A set of current tree nodes

Output: A list of feasible tuples

1. Determine if we can prune the current set of nodes.
2. If we cannot prune the nodes:
 3. If all of the nodes are leaf nodes:
 4. Explicitly test each combination of the nodes' points for feasibility (using one point per node), adding each new feasible set to the results.
 - else
 5. Let X be the non-leaf node from the current set that owns the *largest* number of points.
 6. Recursively call Multiple Tree Track Initiation, using X 's right child in place of X .
 7. Recursively call Multiple Tree Track Initiation, using X 's left child in place of X .

Figure 7: The recursive algorithm for multiple tree track initiation.

The use of multiple trees has previously been explored for efficiently answering spatial queries [5, 4]. However, this work has so far been restricted to simple spatial proximity queries on points. Hjalton and Samet introduced a multiple tree method to efficiently answer spatial join queries in databases [5]. Gray and Moore present a multiple tree algorithm for statistical N-body problems such as spatial correlation functions [4]. Our work extends the multiple tree approach to a significantly more complex question that considers whether points could lie along a trajectory of a given form. This extension results in a more involved pruning query. Efficiently answering this pruning query is a key contribution of this work and is discussed in detail in Section 5.

5. PRUNING

In order for the above algorithm to be effective, we need to accurately and efficiently prune infeasible sets of nodes. With each node we store a bounding box enclosing the points that it owns. Pruning is equivalent to asking: "Can there exist *any* track that passes through *all* M bounding boxes?" 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 model parameters $(\mathbf{a}, \mathbf{b}, \mathbf{c})$ that satisfy the constraints imposed by each node or proving that such parameters do not exist. The bounding box of the i th tree defines $2D$ constraints on what qualifies as a feasible track, an upper bound \mathbf{H}_i and lower bound \mathbf{L}_i on the track's position at that time. In the quadratic case, the constraints on dimension d are:

$$\begin{aligned} \mathbf{a}[d] \cdot t_i^2 + \mathbf{b}[d] \cdot t_i + \mathbf{c}[d] &\leq \mathbf{H}_i[d] \\ \mathbf{a}[d] \cdot t_i^2 + \mathbf{b}[d] \cdot t_i + \mathbf{c}[d] &\geq \mathbf{L}_i[d] \end{aligned} \quad (4)$$

For simplicity we do not explicitly include the thresholds δ^L and δ^H , but rather assume that these are accounted for in the nodes' bounds. Thus we wish to find a vector $(\mathbf{a}, \mathbf{b}, \mathbf{c})$ that satisfies all $2MD$ constraints. It is important to note that these constraints are *exactly* the same ones that we would have to satisfy in order to check the feasibility of a given set of observations. Specifically, δ^L and δ^H define small bounding boxes centered on each of the observations. Thus, pruning a node in our tree search is an equivalent operation to testing a set of observations under an exhaustive approach.

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 i th 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 i th dimension for all $1 \leq i \leq D$. \square

Theorem 1 allows us to consider each dimension separately, reducing the pruning query with $2MD$ constraints to D sub-queries of $2M$ constraints. Further, the separation means that each sub-query consists of significantly fewer variables. For example, in the case of quadratic models 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 models, the results and discussion can be applied to models of other forms. However, the actual computational cost of pruning will vary with the model complexity.

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. For example, in the case of one dimensional quadratic tracks, the constraint for upper bound h :

$$a \cdot t^2 + b \cdot t + c \leq h \quad (5)$$

forms a plane in the 3-dimensional parameter space (a, b, c) :

$$a \cdot (t^2) + b \cdot (t) + c - h = 0 \quad (6)$$

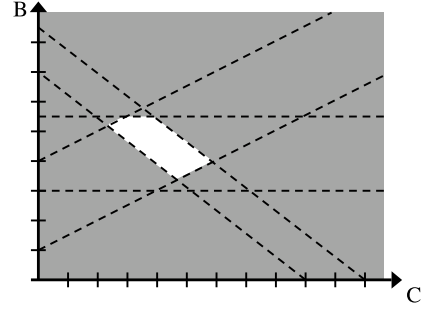


Figure 8: Tracks that conform to all of the constraints lie in the unshaded region of parameter space that is defined by the intersection of the constraint half-spaces.

with normal $(t^2, t, 1)$. Note that the curve's parameters $(a, b, \text{ and } c)$ define the parameter space while node's bounds $(t^2, t, \text{ and } h)$ define the actual plane. Further, each constraint 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 for all of the constraints 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 8. 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:

THEOREM 2. The intersection of M half-spaces defined by at least C nonparallel 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 M 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 M 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. \square

Theorem 2 does not require that the feasible region is fully enclosed by the hyper-planes. Instead, it only requires that there ex-

Brute Force Search

Input: A set of current tree nodes.

Output: A Boolean indicating whether we can prune the search.

1. $\text{prune} \leftarrow \text{TRUE}$
 2. FOR each dimension d :
 3. Let \mathbf{Y} be an empty set of constraint hyper-planes.
 4. FOR each node x in the current set of nodes:
 5. Add the constraint hyper-plane for x 's upper bound in dimension d to \mathbf{Y} .
 6. Add the constraint hyper-plane for x 's lower bound in dimension d to \mathbf{Y} .
 7. FOR each set of k nonparallel hyper-planes in \mathbf{Y}
 8. $\text{valid} \leftarrow \text{TRUE}$
 9. $\mathbf{z} \leftarrow$ the point of intersection of the k hyper-planes
 10. FOR each hyper-plane $y \in \mathbf{Y}$
 11. IF \mathbf{z} is not compatible with y
 12. $\text{valid} \leftarrow \text{FALSE}$
 13. IF $\text{valid} == \text{TRUE}$
 14. $\text{prune} \leftarrow \text{FALSE}$
 15. RETURN prune
-

Figure 9: The brute force search for testing the feasibility of a set of nodes. For each dimension we test all of the corners, looking for one that is compatible with all of the constraints.

ists at least C nonparallel hyper-planes. Under this condition there will be at least one corner to search and the theorem holds. 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, shown in Figure 9. Specifically, 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 $O(M^{C+1})$ operations: $O(M)$ feasibility tests for each of the $O(M^C)$ corners. Below we discuss how we can use the problem's structure 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:

$$\begin{aligned} \mathbf{a}[d] \cdot \mathbf{t}_i^2 + \mathbf{b}[d] \cdot \mathbf{t}_i + \mathbf{c}[d] &\leq \mathbf{H}_i[d] \\ \mathbf{a}[d] \cdot \mathbf{t}_i^2 + \mathbf{b}[d] \cdot \mathbf{t}_i + \mathbf{c}[d] &\geq \mathbf{L}_i[d] \end{aligned} \quad (7)$$

5.2 Using Structure in the Search

In general the above brute force search is too expensive to be used in pruning. We can mitigate this cost by exploiting structure inherent in the tree search.

At each level of the search, the constraints for all tree nodes except one are identical to the previous level. This observation follows from the fact that we are only splitting one node at each level of the search. All of the other nodes and their bounds remain unchanged. Thus we can avoid the computation altogether if the feasible track found at the previous level is compatible with the few new constraints. We can cache the most recent feasible track and test it against the new constraints. If it remains compatible, we do not need to resolve all of the constraints.

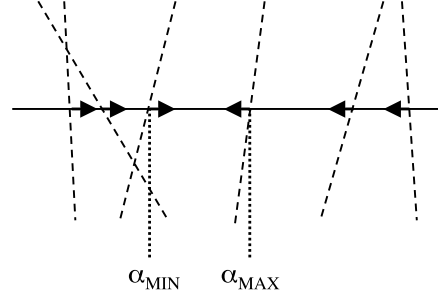


Figure 10: We can check for feasibility along a line by checking the signed intersections of each constraint and the line.

At each level of the search, the constraints for the one tree node that changed are tighter than at the previous level. This observation follows from the fact that each time we split a node, the bounds of the children nodes are contained within the bounds of the parent node. We can use this observation to limit the number of corners that we need to check. Specifically, if the old point is not compatible with a given 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:

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}' . \square

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 $O(M^{C+1})$ to $O(M^C)$.

We can combine the search step and the checking step. 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 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 at a signed point on the line. We can examine all of these signed intersections, tracking the maximum in each direction and asking whether *any* point on the line is feasible. An example is shown in Figure 10. This joint searching and checking step means that we only need to check $O(M^{C-1})$ lines instead of $O(M^C)$ corners.

By using the structure within the search we are able to reduce the cost of a pruning query from $O(M^{C+1})$ to $O(M^{C-1})$. While this may not appear significant, for five time steps ($M = 5$) these

Smart Brute Force Search

Input: \mathbf{Y} - the previous set of constraint hyper-planes.
 \mathbf{x} - the previous feasible point.
 \mathbf{Z} - a set of new constraint hyper-planes.
Output: \mathbf{Y}' - the new set of constraint hyper-planes.
 \mathbf{x}' - the new feasible point (or NULL if the no feasible point exists).

1. $\mathbf{x}' \leftarrow \mathbf{x}$
 2. $\mathbf{Y}' \leftarrow \mathbf{Y}$
 3. FOR each new constraint $z \in \mathbf{Z}$
 4. Add z to \mathbf{Y}' replacing the corresponding constraint
 5. IF \mathbf{x}' is not compatible with z
 6. FOR each set of $k-2$ nonparallel $y \in \mathbf{Y}'$
 7. Let L be the line of intersection of the $k-2$ hyper-planes and z .
 8. valid \leftarrow TRUE
 9. FOR each $y \in \mathbf{Y}'$ that is parallel to L
 10. IF L is not on the correct side of y
 11. valid \leftarrow FALSE
 12. FOR each $y \in \mathbf{Y}'$ that is not parallel to L
 13. Compute the point and direction of intersection of L and y
 14. Let α_{MIN} be the front most point of intersection for planes pointing backwards along the line L .
 15. Let α_{MAX} be the front most point of intersection for planes pointing forwards along the line L .
 16. IF (valid == TRUE) \wedge ($\alpha_{MIN} \leq \alpha_{MAX}$)
 17. $\mathbf{x}' \leftarrow$ the point along L at $(\alpha_{MIN} + \alpha_{MAX})/2$
 18. IF no new feasible \mathbf{x}' was found
 19. $\mathbf{x}' \leftarrow$ NULL
 20. return \mathbf{x}' and \mathbf{Y}'
-

Figure 11: The “smart” brute force search for testing the feasibility of a set of nodes in one dimension. This search is called with the set of constraint hyper-planes \mathbf{Y} and the feasible point \mathbf{x} from the previous level. This search is run for each dimension at each level of the search.

improvements reduce the cost of each pruning query by a factor of 25. The full pruning test is shown in Figure 11.

5.3 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 an object can undergo. These additional constraints can be specified directly:

$$\begin{aligned} \mathbf{a}[d] &\leq \mathbf{a}^{MAX}[d] \\ \mathbf{a}[d] &\geq \mathbf{a}^{MIN}[d] \end{aligned} \quad (8)$$

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

5.4 Missing Observations

Up to this point, our discussion has assumed that each object produces one observation *every* time step. There are several simple approaches to handling missing observations. Perhaps the easiest is to include “missing” as a single new node in the tree as shown in Figure 12. Additional logic can be added to prune the search if too

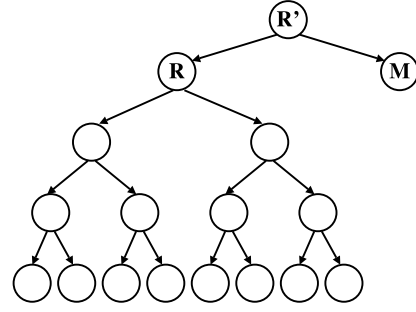


Figure 12: One approach to handling missing points is to treat “missing” as another tree node. Here the new tree is shown as a new root \mathbf{R}' and a missing node \mathbf{M} .

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.

5.5 Unknown or Complex Models

The above discussion assumes that we have a known and relatively simple model. However in many domains, this may not be the 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 a track to undergo retrograde motion.

A poor or unknown model can easily be approximated with a simple model. However, 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, it is possible to use the above algorithms to find short arcs that can then be glued together by a conventional tracking algorithm. Such an approach was suggested by Shaw and Arnold [10].

6. EXPERIMENTS

6.1 Algorithms

In examining the performance of the multiple tree approach, we compared the range of approaches from a sequential approach to a full multiple tree approach. Specifically, we used multiple tree algorithms, denoted MT-1 through MT-M, with the following descent rule (replacing line 5 of the algorithm in Figure 7):

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 time steps.

When $k = 1$, this algorithm is a kind of sequential track initiation algorithm that uses our feasibility constraints and KD-trees. 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 that are compatible with the tentative track. Pruning is only done in relation to whether the trees traversed so far allow a

valid track. It is important to note that this approach uses our feasibility constraints and thus it is still an exact algorithm for finding the desired linkages. 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 also important to note that all versions of this descent rule use the same feasibility criteria. Thus all variations are exact algorithms and will return the same set of associations.

In the below experiments the KD-trees for all of the algorithms were constructed in an identical manner. The points were recursively partitioned by splitting on the midpoint of the widest dimension. The partitioning continued until a node had only one point or its bounding box had zero width. The decision to build the tree down to a single point per leaf arises naturally from the fact that our pruning criteria is identical to the feasibility test and that we are able to exploit structure from the recursive nature of the search to accelerate the feasibility test. Specifically, it should be noted that the pruning test for a set of leaf nodes exactly answers the feasibility test for the points owned by these leaf nodes.

6.2 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:

$$\begin{aligned} \mathbf{a} &\sim \text{uniform}(-\mathbf{a}_{\text{MIN}}, \mathbf{a}_{\text{MAX}}) \\ \mathbf{b} &\sim \text{uniform}(-\mathbf{b}_{\text{MIN}}, \mathbf{b}_{\text{MAX}}) \\ \mathbf{c} &\sim \text{uniform}(0, 1) \end{aligned} \quad (9)$$

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 :

$$\mathbf{x}_i^t[d] = \frac{1}{2} \mathbf{a}_i[d] t^2 + \mathbf{b}_i[d] t + \mathbf{c}_i[d] + \epsilon \quad \forall d : 1 \leq d \leq D \quad (10)$$

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

Below we describe the relative performance of the descent rules. Again, it is important to note that the MT-1 descent rule is effectively a sequential algorithm with our feasibility criteria and with KD-trees. In addition, we computed the cost that would be incurred by an exhaustive algorithm (*EXH*) that considers all M -tuples of observations.

6.2.1 Effect of the Number of Objects

The primary factor that we would expect to influence the performance of the algorithms is the number of objects. We varied the number of objects from 50 to 5000 and compared the number of pruning tests. 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 associations appear feasible. The effect of density is illustrated in Table 2. This table 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 aids all algorithms, but especially helps the multiple tree algorithms.

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

N	EXH	MT-1	MT-2	MT-3	MT-4
50	3.1×10^8	2.9×10^3	2.5×10^3	2.3×10^3	2.6×10^3
100	1.0×10^{10}	7.6×10^3	6.9×10^3	6.2×10^3	8.4×10^3
500	3.1×10^{13}	1.1×10^5	1.0×10^5	8.4×10^4	1.5×10^5
1000	1.0×10^{15}	4.7×10^5	4.5×10^5	3.3×10^5	5.8×10^5
5000	3.1×10^{18}	2.6×10^7	2.6×10^7	2.1×10^7	2.2×10^7

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

N	EXH	MT-1	MT-2	MT-3	MT-4
50	3.1×10^8	2.5×10^3	2.1×10^3	1.7×10^3	1.4×10^3
100	1.0×10^{10}	5.7×10^3	4.8×10^3	4.0×10^3	3.3×10^3
500	3.1×10^{13}	3.9×10^4	3.2×10^4	2.6×10^4	2.2×10^4
1000	1.0×10^{15}	8.7×10^4	7.2×10^4	6.0×10^4	5.4×10^4
5000	3.1×10^{18}	6.1×10^5	5.2×10^5	4.2×10^5	4.9×10^5

6.2.2 Effect of the 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 association is relatively well confirmed and additional trees do not help.

Table 3: Average number of pruning tests (in millions) for linear models ($\mathbf{b}_{\text{MAX}}[d] = 0.1 \forall d \geq 1$) with 5000 observations at 5 times with varied spacing. Note that an exhaustive algorithm would have required 3.125×10^{18} tests regardless of the spacing in time.

Δt	MT-1	MT-2	MT-3	MT-4	MT-5
0.01	0.52	0.42	0.32	0.23	0.15
0.05	0.58	0.49	0.39	0.39	0.44
0.10	0.76	0.67	0.49	0.60	0.86
0.50	5.30	5.75	1.97	3.95	8.18
1.00	20.80	20.60	5.52	11.30	22.70

Table 4: The number of pruning tests (in millions) for the astronomy data with various numbers of time steps.

Times	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

6.2.3 Comments and Discussion

It is interesting to note that while the use of multiple trees can provide a computational benefit, as the number of trees grows this benefit may be negated or even turn into a detriment. For example, the experiments in Section 6.2.1 and Section 6.2.2 show a superior performance of MT-3 relative to MT-4. The use of multiple trees provides a trade off between providing additional pruning information from later time steps and introducing a higher branching factor to the search. More trees at later time steps introduce more opportunities for “wrong turns” in the search and require a deeper search to limit the regions covered by each node. It is our current belief that this effect accounts for MT-3’s relative performance. The use of three trees provides enough information to estimate and initially confirm a track, without forcing us to backtrack the search to test many further support points. In general, we would expect the optimal number of trees to depend on both the complexity of the track model and the distribution of the data. This effect has interesting implications for multiple tree algorithms and we are continuing to examine it as future research.

6.3 Astronomy Data

We also examined simulated data from the astronomy domain in order to test whether this algorithm provides a benefit on objects 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 model over a period of 16 nights and observations were generated from this model. Each observation consisted of two components, right ascension and declination, that gave the object’s projected 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 at different numbers of time steps 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.

It is important to note that the actual running times of the algorithms are proportional to the number of pruning queries. For example, on the runs with 6 time steps MT-1 required 274 seconds while MT-3 required 204 seconds on a 1.25 GHz Apple G4 with 512 MB of RAM. Further, the relative speedups are independent of processor dependent accelerations and code optimizations.

7. CONCLUSIONS

Above we describe an exhaustive methodology for finding sets of points that are compatible with a given type of motion model. We motivated and presented these techniques in the context of track initiation and asteroid linkage. We introduced a multiple tree algorithm for tractably finding the linkages. Further, we presented an efficient pruning methodology for this search that uses structure

from both the feasibility problem and the search itself. Empirically, this algorithm performed very well on several simulated data sets, outperforming an exact adaptation of conventional multiple hypothesis tracking.

The true advantages of the multiple tree algorithm are its ability to use information from later time steps to aid in pruning decisions at earlier time steps and to avoid repeated work over similar sets of points. 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 increases and bounds on model parameters (such as maximum velocity) become weaker.

8. ACKNOWLEDGMENTS

Jeremy Kubica is supported by a grant from the Fannie and John Hertz Foundation. Andrew Moore and Andrew Connolly are supported by a National Science Foundation ITR grant (number CCF-0121671).

9. REFERENCES

- [1] Y. Bar-Shalom and X. Li. *Multitarget-Multisensor Tracking: Principles and Techniques*. Yaakov Bar-Shalom, 1995.
- [2] J. Bentley. Multidimensional binary search trees used for associative searching. *Comm. of the ACM*, 18(9), 1975.
- [3] S. Blackman and R. Popoli. *Design and Analysis of Modern Tracking Systems*. Artech House, 1999.
- [4] 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.
- [5] G. R. Hjaltason and H. Samet. Incremental distance join algorithms for spatial databases. In *Proceedings of the 1998 ACM-SIGMOD Conference*, pages 237–248, 1998.
- [6] P. V. C. Hough. Machine analysis of bubble chamber pictures. In *International Conference on High Energy Accelerators and Instrumentation*. CERN, 1959.
- [7] D. Reid. An algorithm for tracking multiple targets. *IEEE Transactions on Automatic Control*, 24(6):843–854, December 1979.
- [8] V. Salari and I. K. Sethi. Feature point correspondence in presence of occlusion. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 12(1):87–91, January 1990.
- [9] I. K. Sethi and R. Jain. Finding trajectories of feature points in monocular image sequences. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-9:56–73, 1987.
- [10] S. W. Shaw and J. 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.
- [11] J. K. Uhlmann. Algorithms for multiple-target tracking. *American Scientist*, 80(2):128–141, 1992.
- [12] 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.
- [13] C. Veenman, E. A. Hendriks, and M. Reinders. A fast and robust point tracking algorithm. In *Proceedings of the Fifth IEEE International Conference on Image Processing*, October 1998.