Methods for Solar System Object Searching in "Deep Stacks"

by

Jonathan Myers

A Dissertation Submitted to the Faculty of the

Department of Computer Science

In Partial Fulfillment of the Requirements
For the Degree of

Masters of Science

In the Graduate College

The University of Arizona

2 0 0 8

Table of Contents

TABLE OF CONTENTS—*Continued*

List of Figures

List of Tables

Abstract

Forthcoming surveys of the solar system will come from next-generation telescopes such as PanSTARRS and LSST. For most cases, current multiple-hypothesis tracking (MHT) methods are a computationally-efficient and reliable method for tracking and identifying solar system given source-extracted observed points from images of the sky. However, the variety of science goals for these instruments will result in certain observing cadences which generate data intractable for current MHT approaches.

In particular, we focus on "deep stacks" of images in which a single area of the sky is imaged many times in rapid succession. These "deep stack" collections are a poor match for existing multiple hypothesis tracking methods, resulting in exponential growth in number of hypotheses.

This thesis presents and evaluates new heuristic methods and tools for efficiently extracting usable tracks (linkages between observed points) from "deep stacks", allowing "deep stacks" to be used with existing MHT-based tools.

Though our application in this case is on solary system object search, these methods could potentially be adapted to other problems in computer vision or machine learning, in which one wishes to identify an unknown number of objects moving according to a linear motion model given a large set of noisy but sparse data points. Similarly, many of the methods and filters presented here could be applied to any multiple hypothesis tracking method.

Chapter 1

Introduction

1.1 Objective

Future surveys of the solar system will be dependent upon the ability to automatically process large amounts of data in the form of source-extracted points and corresponding magnitudes. These systems will be responsible for determining the number and behavioral parameters of observed solar system objects such as comets and asteroids, and with correctly attributing observed points to their source objects (or to noise).

The resulting problem of generating an initial set of possible object orbits and attributions has been approached with considerable success using multiple hypothesis tracking (MHT) methods based on spatial data structures [Kubica, 2005]. These methods take advantage of various assumptions about the data, including the behavior and observing schedule (or *cadence*) of the observing instrument.

This thesis deals with the problem of solar system object searching in the context of *deep stacks*, that is, collections of many images of the same (or similar) areas of the sky over a short period of time, as they will be observed by the LSST[Ivezic et al., 2008] and Pan-STARRS[Kaiser, 2004] telescopes. This case is a poor fit for existing MHT methods, resulting in exponential computation and storage times.

The methods presented and analysed in this thesis are a substitute to the initial steps of the existing multiple hypothesis tracking methods, intended to effectively identify probable hypothetical linkages in a deep stack of images. This allows the existing methods to be applied to a larger set of input sources.

In the terminology of asteroid search, we are concerned with *tracklet* generation: the problem of proposing sets of observations which follow a linear path. These intranightly linkages between observations are then linked together to form longer *tracks* which span periods up to one lunation.

Tracklets are a tool used within target tracking for various applications ([Drummond et al., 2003]), and the methods here could feasibly be applied outside the field of solar system object tracking.

Generally, this thesis deals with the problem of finding associations between apparitions of the same underlying object, in the special case where the number of underlying objects is unknown, the apparitions are sparse but hidden among noisy data and the underlying motion is linear or nearly-linear.

1.2 Background

This section is a brief introduction to some of the basic ideas related to solar system object search in general, and is intended to provide a broader context for a non-astronomy reader. Chapter 2.1 presents the existing search system, which is considerably more important for understanding the new work presented in this thesis.

1.2.1 An Introduction to Solar System Astronomy Concepts

Celestial Motion The mechanics of 2-body celestial motion are still expressed as they were by Kepler, 1605. The motion of asteroids, comets, planetoids and planets can be described with Keplerian orbits, which describe orbital motion with six parameters:

eccentricity

semimajor axis

inclination

longitude of the ascending node

argument of periapsis

mean anomaly at epoch

However, in this thesis we deal mainly with the single-night case, during which the apparent motion of an object through the sky is almost exactly linear[Kubica, 2005].

Topocentric Coordinates When we discuss points on the sky, we do so in terms of *topocentric coordinates.* Topocentric coordinates express locations on the Earth's sky in terms of angles in latitude and longitude offset from a known point, the vernal equinox.

Ephemeris Generation The *observations* or *apparitions* of an object are generally expressed as topocentric points on the sky with magnitude information. When we attempt to predict the points which of these point sources, this is called *ephemeris* generation. This is expressed with the following equation:

$$\psi = \Psi(P) + \epsilon + \nu \tag{1.1}$$

Where ψ is a set of coordinates with magnitude, Ψ is the *ephemeris function*, P is an expression of the six orbital elements, ϵ expresses random error and ν expresses systematic error[Granvik, 2007].

Orbit Determination Ultimately, asteroid search methods wish to extract orbital fits for observed objects. With six variables for which to solve, it is generally understood that six observed points are needed, though additional points will help minimize error. It is generally considered preferable for observations to span several days, as this will allow underlying motion to become more significant relative to observational error. For a detailed treatment of this problem, see Granvik [2007]. It is important to note that this is a fairly computationally expensive process, as we (conceptually) must choose the one orbit out of an infinitely large number which would generate the observed points with minimal error. Additional complications are possible, as for a set of observations, there may be several probable, but extremely different, orbits Milani et al. [2006].

1.3 Relevant Telescope Projects

1.3.1 LSST

The LSST (Large Synoptic Survey Telescope) is a planned Earth-based telescope with an 8.4 meter diameter aperture, a 9.6 degree-squared field of view, and a 3.4 Gigapixel camera. LSST goals will be extremely diverse, from identifying asteroids (including NEOs, and Kuiper-belt objects) and comets to detection of weak lensing and catching transient events like supernovae [Ivezic et al., 2008]. LSST's image scheduler will attempt to visit each region of sky it images twice per night with a 30 to 90 minute delay between images, with a per-lunation cadence intended to be amenable to asteroid search (i.e., revisiting asteroid-dense areas on three nights per lunation, with at least two observations per region per night.) However, the diverse science goals of the LSST will mean that many regions of the sky will be revisited more than twice per night, resulting in many "deep stacks" up to one hundred images of the same region. Investigating an ideal cadence is an ongoing process carried out by the LSST Operations Simulator team; for a detailed treatment, see Cook et al. [2009].

1.3.2 PanSTARRS

The PanSTARRS (Panoramic Survey Telescope and Rapid Response System) is another next-generation Earth-based telescope project intended to fulfill a significant subset of the LSST science goals. PanSTARRS PS4 is a planned "distributed-aperture" telescope, consisting of an array of four coordinated telescopes with 1.8 meter diameter mirrors and a 7 square degree views of the sky[Kaiser, 2004]. PanSTARRS PS1, the single-telescope prototype, is currently (as of this writing) active atop Mount Haleakala. PanSTARRS hopes to find most near-Earth objects (NEOs) down to about 300 meters in diameter as a primary goal[Kaiser, 2004]; hence the cadence is highly conducive to asteroid survey. In the general case, on a given evening, a variety of regions of the sky will be visited, and each region will be visited twice with a time between observations of 30 to 90 minutes. Most regions will be visited on three nights per lunation for a total of six images per region. However, to sample light curve data from asteroids, some regions may be visited up to one hundred times in "deep stacks."[Jedicke, 2008].

Chapter 2

An Overview of Asteroid Search

2.1 An Overview of the Moving Object Pipeline System

The Moving Object Pipeline System, or MOPS, is the name of the object identification system used by PanSTARRS and LSST.

The input to the MOPS is a set of *detections*: individual point sources which may be moving objects like asteroids or comets, or may simply be noise (incidental cosmic rays, artifacts of image processing, etc.)

The job of the MOPS is to find sets of possible object *orbits* which *may* explain some of the detections. Note that it is generally preferable to report slightly too many orbits; it is far better to have a false positive (which can be disproven later) than to miss an object.

Since we expect on the order of thousands of detections per image, and perhaps hundreds of images per night, it is necessary to use intelligent structured search in order to find these orbits.

The components of the MOPS which we examine deal with finding *sets of detections* which *may* belong to the same object. These sets are often called "linkages" as they "link" together detections (or other structures).

2.2 Stages of the MOPS System

2.2.1 Tracklet Generation

The Tracklet Generation phase takes raw detections as an input, and produces linkages between pairs of detections, which are called *tracklets*. The term *tracklet* is generally used in the literature, particularly for target tracking, as a small set of associated data items.

In the case of the MOPS, a tracklet is (generally) a linked pair of detections from the same night. Since the objects are moving, the tracklets have an implied linear motion which can be used to help later searching (as we shall see).

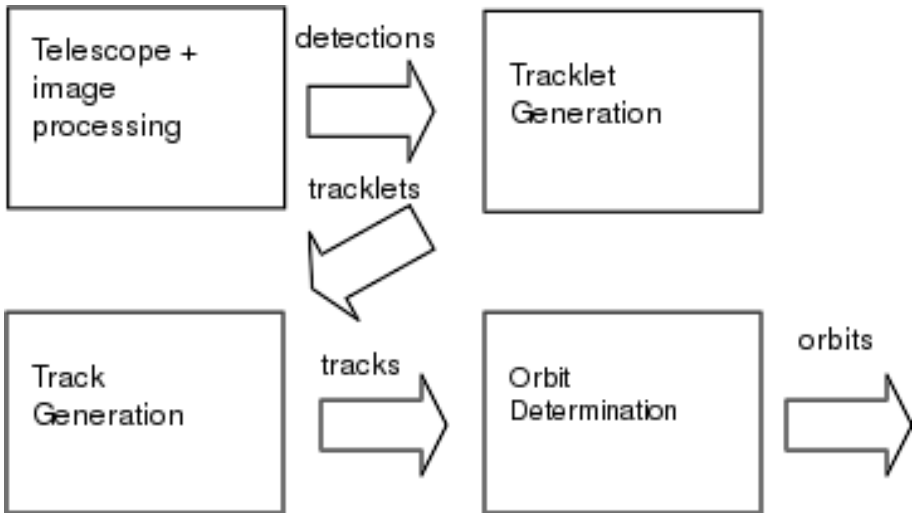

FIGURE 2.1. An illustration of the overall MOPS system. Note that each phase uses the prior stage as input. In a sense, each stage "filters" its input, but it always provides a generous amount of data to the next stage - otherwise, an orbit may be missed.

As with all the other stages, tracklet generation is expected to produce a superset of the "needed" data. That is, we will expect to see incorrect linkages, including linkages between noise and real objects, noise points and noise points, multiple objects, etc. Also note that **there is no constraint on the number of linkages per individual detection.** It is fully expected that **some detections will belong to multiple tracklets**.

Note, however, that since we can place constraints on the maximum (or minimum) observed speed of a moving object across the sky, we can constrain the set of reported tracklets.

2.2 presents an example of tracklets generated from a pair of images.

2.2.2 Track Generation

In the Track Generation (or Tracklet Linking) phase, tracklets are used as input, and longer *tracks* are used as output.

In the case of solar system objects, the path across the sky can be roughly quadratic [Kubica, 2005] over the course of a month. Thanks to some clever algorithms presented in Kubica [2005], track generation can be done fairly quickly using 4dimensional KD-Trees which hold tracklets represented by location and velocity in the two

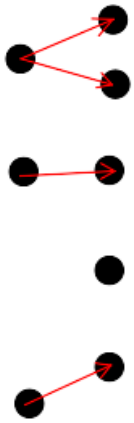

FIGURE 2.2. An illustration of tracklets generated from a pair of images, simplified down to two dimensions. Note that some detections have multiple associated tracklets. Some detections have no associated tracklets, perhaps because there was no second detection which was close enough to be a feasible match.

axes of right ascension and declination.

After linking, a tracklet may belong to zero or more tracks. Some tracklets will belong to many tracks. 2.3 illustrates a pair of tracks which share one tracklet.

The illustration 2.4 shows a simplified example of the MOPS system up to this point.

2.2.3 Orbit Determination

Orbit Determination is the process of approximating an object's orbit around the sun based on its observed locations on Earth's sky.

In the Orbit Determination phase, we attempt to derive an orbit for each track's set of associated detections.

Note that the orbit determination phase can cull out many incorrect tracks, as they will generally have no sufficiently tight-fitting orbit. However, the set of orbits which is produced is expected to be **a superset** of the actual set of orbits. A "false alarm" can be identified later when the hypothetical object does not appear as expected; this is a separate problem.

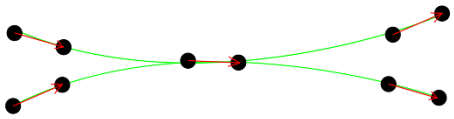

FIGURE 2.3. An illustration of a tracklet which belongs to multiple quadratic tracks.

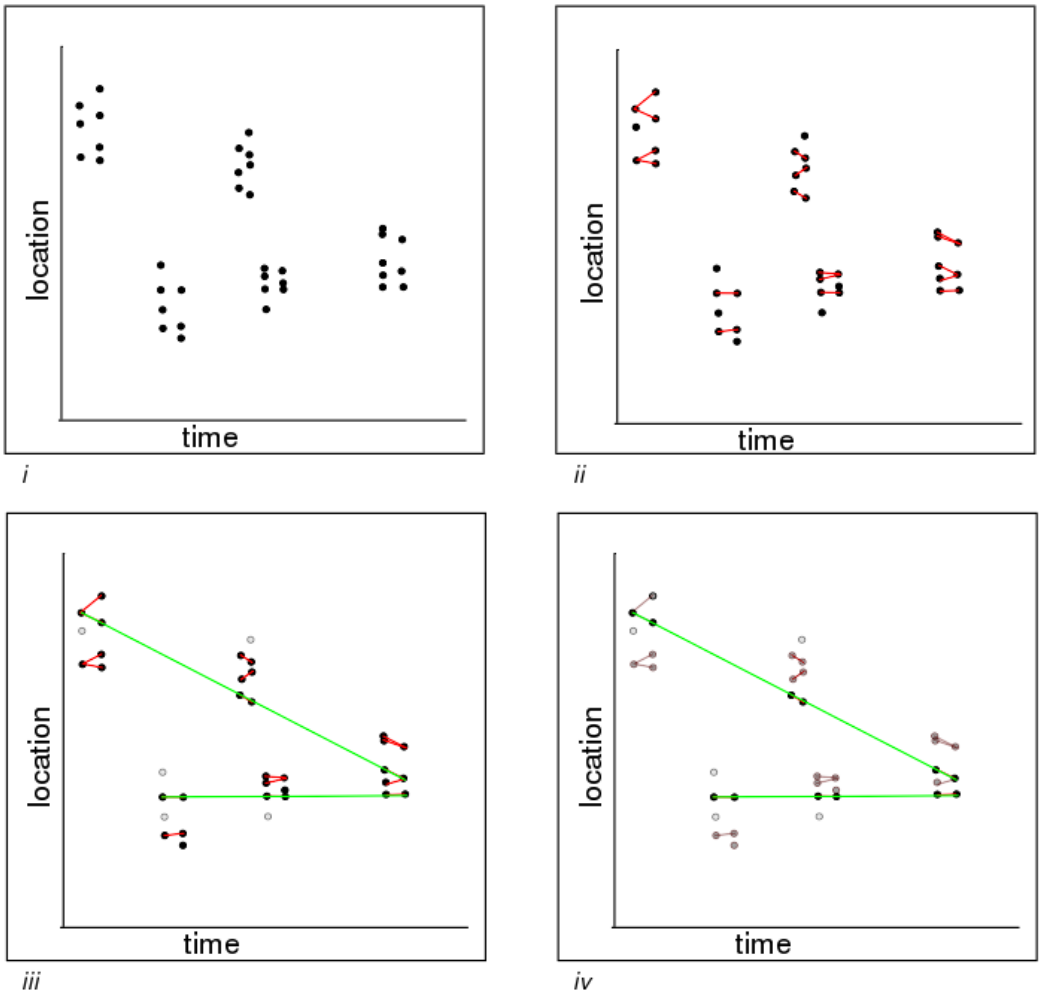

FIGURE 2.4. An illustration of the MOPS system, simplified to one dimension of location and one dimension of time. In *i*, we see a series of detections which originate from pairs of images. In *ii*, pairs of nearby detections in each stack are joined as "tracklets" (drawn in red). Note that individual detections may belong to multiple tracklets. In *iii*, tracklets are linked into tracks, drawn in green. Those detections not associated with a tracklet are forgotten, and thus are drawn in gray. Finally, in *iv*, all tracklets not associated with a track are forgotten (and here are drawn in gray). The resulting tracks are passed to an orbit determination tool.

Chapter 3

The Problem of Tracklet Generation from Deep Stack Data

3.1 2-image Stacks Versus "Deep Stacks"

In the Tracklet Generation phase of the MOPS, tracklets are usually generated from *pairs* of images from the same region of the sky, which are generally taken between 15 and 90 minutes apart. The choice of image locations is called *cadence* and in Pan-STARRS and LSST, the telescope cadence is chosen to satisfy many factors, of which moving object search is only one.

This thesis is concerned with allowing the MOPS to intelligently and quickly handle "deep stacks:" sets of many images from the same night, at the same location of sky. As we shall see, this introduces some interesting new complications.

Normally, in a two-image stack, the number of tracklets and the time to find them is fairly low. Given p detections in the first image and q in the second, the upper bound on the possible number of tracklets is $p * q$ (and of course, in practice the number is much lower).

However, in a "deep" stack, it is not entirely clear what set of tracklets should be reported.

3.2 Full Enumeration of All Tracklets In A Deep Stack

As described in 2.2.1, the set of tracklets produced in the tracklet generation phase is used as the input to track generation phase. Since the track generation phase treats tracklets as "atomic," and as its only input, if a tracklet which is needed for generating a track is not present, that track is never generated.

From this perspective, one "safe" approach to ensuring that all needed tracklets are represented would be to produce all possible tracklets. We shall refer to this as "pure" multiple hypothesis tracking (MHT) approach: if the tracklets are hypothetical portions of tracks, we present *all* hypotheses to the track generation phase.

This section will explain why this idea is **not** tractable, nor is it practical.

3.2.1 Computational Challenges On Pure MHT Tracklet Generation

Consider m mutually compatible detections in a deep stack - that is, m observed points which fall roughly along the same line. A Pure MHT tracklet generation scheme would require us to report every possible set of linkages between these detections - including linkages of 2 detections, linkages of 3 detections, linkages of 4 detections, etc., and the linkage of all m detections.

There will be $m \times (m-1)$ 2-linkages, $m \times (m-1) \times (m-2)$ 3-linkages, $m \times (m-1) \times (m-2) \times (m-3)$, and so on. In fact, we will enumerate *the power set* of the set of compatible detections. As a result, there will be $O(2^m)$ tracklets as a product of *just* these m mutually compatible detections!

Of course, a real image stack will likely contain many mutually compatible detections; if our configuration is correct, then every observed detection of the same real object should be mutually compatible! As a result, pure MHT is a poor choice. Given a stack of 64 images, and just one object visible in all 64, pure MHT would generate 2^{64} tracklets - more than a 64-bit computer could even address in memory!

Figure 3.1 shows an example illustration of four compatible points from a single night of observation; Table 3.1 shows the set of all physically plausible linkages arising from this example.

3.3 Problem Formulation

Pure MHT seeks to find *all* possible sets of linkages such that the detections linked are mutually compatible with the motion of an underlying object.

As this is infeasible in the case of a deep stack, we instead attempt to present an alternative approach: to report a *subset* of the total set of feasible tracklets, such that the *correct* tracks can be constructed from these tracklets.

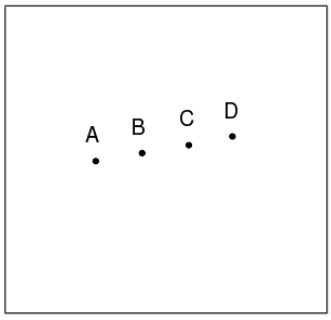

FIGURE 3.1. Illustration of four compatible detections, as one would expect from four apparitions of the same object in a single night. See 3.1 for a set of all possible resulting linkages.

2-linkages	3-linkages	4-linkages
A B	A B C	A B C D
A C	A B D	
A D	A C D	
B C	B C D	
B D		
C D		

TABLE 3.1. The set of all physically plausible tracklets (linkages) for the object illustrated in figure 3.1, as would be reported by pure MHT methods for tracklet generation. The tracklets are organized by number of linkages for readability reasons. Note that the number of possible tracklets grows like n^2 for n detected apparitions of the same object; each additional detection will increase the number of possible tracklets by a factor of 2.

Chapter 4

A Suboptimal Solution

4.1 Introduction

As shown in 3, enumerating all possible tracklets is an infeasible solution to the tracklet generation problem in a deep stack.

In this section, we present another solution which is tractable, but far from optimal, and explain why it is insufficient.

4.2 Reporting Length-2 Tracklets Only

4.2.1 Benefits

Say that even in a deep stack, we were to report only the two-detection tracklets: that is, just as in a two-image stack, we report only linkages between exactly two detections. There are several reasons why this approach is appealing, at least compared to Pure MHT.

Findable Objects Remain Findable Given an object which is observed at least twice, there will exist at least one tracklet which links together two of those detections. This tracklet can be used in tracklet linking and track generation. As a result, we can be confident that this method is "safe:" objects which *should* be "found" and have derived tracks *will* be.

Reasonable, Though not Optimal, Upper Bound On Tracklets Unlike the Pure MHT approach presented in 3, which generates 2^m tracklets for a set of m mutually compatible detections, this method will produce only $m \times (m-1)$, or roughly $O(m^2)$. This is because each detection will be a "starting" point for at least one tracklet, and will be linked with each later detection. If the upper bound on a deep stack is in the hundreds, then the number of tracklets per object is at most less than one million, and in the relatively common case when the number of images in the deep stack is at most twenty or thirty, then the number of tracklets per object is only in the hundreds.

4.2.2 Drawbacks

Despite some benefits, this approach also has significant drawbacks.

Lots of Tracks Consider a real object which has p tracklets from a single deep stack. Assuming the object is observed in enough subsequent images for a track to be generated, then during track generation, there will (under most circumstances) be at least p tracks generated. Unless we employ some cleverness later, orbit derivation will be performed on each of these tracks. All in all, this can put a needless computational burden on the orbit determination phase, which is already quite slow.

Not Very Clever The further intuitive problem here is that if many detections are mutually compatible, it is *highly* likely that they are attributable to the same object! This fact is intuitively obvious even at the tracklet generation level, when we only need to deal with a relatively small amount of data. And by ignoring it we risk missing an easy opportunity to greatly reduce work later, when we deal with enormous amounts of data!

4.2.3 Conclusion and Lessons Learned

This approach demonstrates that by being sufficiently clumsy in the tracklet generation phase, we can greatly increase our computational load later. As a result, the remainder of this thesis will be devoted to the following problem: **how can we be more clever in tracklet generation, reducing our computational load in later stages, with minimal risks to "safety"?**

Chapter 5

Deep Stacks Search Methods

5.1 Introduction

In 4, we presented a scheme for reporting only tracklets which link together exactly two detections. This approach became problematic when an object is detected more than two times; no one tracklet spans all those detections.

In building a better set of tracklets, we will need to find "longer" tracklets: those which contain more compatible detections.

One method would be to use the full MHT method described in 3, and then remove the subset tracklets; however, the incredibly large number of tracklets generated makes this infeasible.

In this chapter, we present a novel heuristic for quickly finding these maximally "long" tracklets. To do so, we exploit the fact solar system objects tend to follow a roughly linear path across the sky.

5.2 collapseTracklets

The collapseTracklets method is a greedy and heuristic tool we developed for quickly finding sets of detections which fall on roughly the same line. Further filters described in this section will refine these tracklets to meet our requirements.

Rough Description Using a KD-tree of all observations, we first find **all** compatible 2-observation linkages (this can be done trivially using existing pure MHT software and correct flags). We then project linearly the hypothetical motion of these 2-observation tracklets out to some common time t_c (e.g. the mean of the earliest and latest observations). We store each tracklet in a KD-tree over the 4-dimensional space of (right ascension at t_c, declination at t_c, angle of hypothetical motion, velocity of hypothetical motion). We then perform a range search on this tree once for each 2-observation tracklet, thus finding tracklets with similar motion. Sufficiently similar tracklets are "collapsed" into longer tracklets.

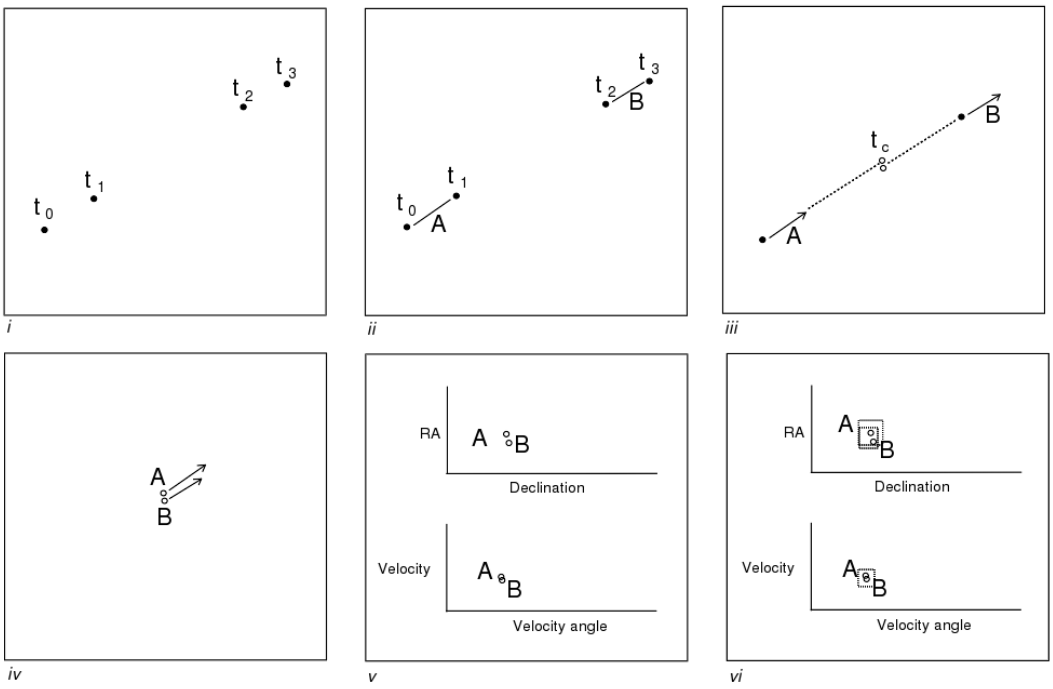

FIGURE 5.1. Illustration of collapseTracklets on four two-dimensional observation points.

An Illustration of CollapseTracklets In 5.1 we see an illustration of collapseTracklets applied to some two-dimensional observations of a single object.

In *i*, points are illustrated and labelled with the time at which they are observed, t_0, t_1, t_2, t_3. Assume that the interval of time between t_0 and t_1 is equal to that between t_2 and t_3, but that the amount of time between t_1 and t_2 is longer.

In *ii*, we see a 2-element subset of the physically plausible 2-observation tracklets (the set of all physically plausible tracklets would include linkages between each pair of points.) These two tracklets are labelled A and B.

In *iii*, we see the apparent velocities of the two tracklets. These velocities are used to estimate the likely location(s) of A and B at a common time t_c.

In *iv*, we see one visual representation of A and B as vectors in the four-dimensional space of (RA, Dec, velocity, velocity angle). The circles represent the RA and Dec at the common time t_c and the arrows represent velocity.

In *v*, we see an alternative representation of this pair of four-dimensional points.

Finally, in *vi*, we see an illustration of a range search around A in (RA, Dec, RA velocity, Dec velocity)-space. B is within the query range, and as a result our four original points will be linked together into a four-observation tracklet $\mathrm{A}' = \mathrm{A} \cup \mathrm{B}$.

5.2.1 collapseTracklets psuedocode

- Let $TInput$ be the set of all physically plausible 2-point linkages between observed points in the set D, arrived at using some other method.
- Choose a time t_c (we use the average of the first and last observation times in D).
- Initialize our output set as $TOutput = \{\}$
- Initialize our tree data as $P = \{\}$.
- For each 2-point tracklet r in $TInput$:
  - Find the apparent velocity of r in right ascension and declination as a function of time.
  - Use the apparent velocity of r to project the topocentric location of r at time t_c; call this point r_{p_c}.
  - Create a four-dimensional point representation of r using r_{p_c} and the velocity and velocity angle described by r's apparent motion. Call this p_r. Add $\{p_r, r\}$ to P. Initialized p_r as "unmarked."
- Build a 4-dimensional KD-Tree using P, with the first element of each $p \epsilon P$ as the spatial data point.
- For each $p \epsilon P$:
  - If p is not yet "marked":
  - Where $p = \{p_r, r\}$, create a new output tracklet r_{out}, containing all point detections associated with r.
    - Use the KD-Tree to perform a 4-dimensional range search centered on p; call the results S.
    - For each $s \epsilon S$ such that s is not "marked":
      - Mark s.
      - Where $s = \{p_s, r_s\}$ add all point detections associated in r_s to r_{out}
    - Add r_{out} to $TOutput$.
    - Mark p.

5.2.2 Execution Time

The input to collapseTracklets is a set of pairs of detections. Technically, there may be as many as n^2 pairs given n detections, but in practice there are roughly $O(n)$ as the data is sparse relative to the limits on the topocentric velocity of real objects; most detections of real objects are only linked with other detections of the same object and a few noise points, and only rarely are they linked with detections of other objects.

CollapseTracklets collapses each input tracklet at most once, and performs tree queries at most once per input tracklet. As tree searches over m data points take time $O(\log m)$, collapseTracklets will take at worst time $O(n^2 \log n)$, and in practice $O(n \log n)$ given n initial detections.

5.2.3 Removing Incompatible Linkages

As collapseTracklets does its linking partly in parameter-space, it is possible that we will "collapse" together detections from tracklets which are similar in their parameter-space representation but not sufficiently linear in the sky-plane. To fulfill our goal of only producing tracklets which contain mutually compatible detections, we will need to remove "bad"linkages.

The most effective approach we have tried removes individual detections from tracklets in a process which we call **tracklet purification**. In tracklet purification, we find the best-fit line to each tracklet's observations. While the RMS distance of the observations to the line is too high, we remove the observation with the highest RMS distance and repeat.

In the worst case, this could be quite painful - if all the elements of a tracklet were mutually incompatible, we would perform one line fit per detection in the tracklet. If all of our tracklets followed this pattern, we would perform

$$\sum_{t \epsilon T} (|t| - 2)$$

line fits, where T is the set of all tracklets and $|t|$ is the number of detections linked together in t.

Fortunately, in practice this processing is quite fast, as it is fairly uncommon for collapseTracklets to return tracklets which contain mutually incompatible detections.

Chapter 6

Tests and Results

6.1 Metrics for Measuring Success and Performance

The measurement of success in tracklet generation is an innately non-trivial problem. Conventional machine learning metrics are only partially applicable, as tracklet generation is merely a prior step to tracklet linking and initial orbit determination, in which the sought data (actual object tracks and object orbits) are found.

6.1.1 Object Coverage

Object coverage attempts to measure the degree to which detections of real objects are conferred to the tracklet linker such that these detections can later be correctly linked with detections from other nights.

Since tracklets are treated as atomic items by the tracklet linking phase, a tracklet which is not "correct" can never be correctly linked with other tracklets to form correct tracks. A tracklet which is not "correct" - one which contains detections from multiple sources - is of no value.

Ergo, **object coverage** for a given object is defined to be the number of unique detections of that object in all correct tracklets divided by the total number of detections of that object.

Ideally, we wish to reach 100% object coverage for each object. In this case, every detection will be capable of being linked into a correct track.

6.1.2 Tracklet Quality

A smaller number of correct tracklets which span many detections are better than a larger number of correct tracklets which span fewer detections, as they will make later linking phases much easier. **tracklet quality** is intended to measure this property.

The quality of a tracklet is defined as the number of detections in the tracklet divided by the number of possible detections of the associated object, provided that the tracklet is correct. If the tracklet is incorrect, it has no usefulness for later linking, and its quality is zero.

Two metrics will be presented: average tracklet quality, the average quality of all tracklets returned, and average quality of correct tracklets, which is the average quality of tracklets which are correct.

Tracklet quality is important; given the same object, a correct track built with a low-quality tracklet will contain fewer detections than a track built with a high-quality tracklet. The track with more detections will result in a better orbit fit. It also associates more of the detections with the object, leaving behind fewer "orphan" detections which would need to be dealt with later.

6.1.3 Tracklet Set Size

This is simply the measure of the total number of tracklets reported. Unlike the prior two metrics, smaller is better, as there will be less total work for the remaining processing steps.

6.1.4 Hypothetical Examples

In the case of the full MHT (3), we would expect high object coverage (as all feasible tracklets are represented, any detection of an object *can* be present in a correct tracklet is present in *some* correct tracklet), mediocre tracklet quality (many high-quality tracklets, many low-quality tracklets) and an abominably large tracklet set size (a huge number of tracklets overall).

Given the set of all two-detection tracklets (as in 4), we would expect equally high object coverage (again, any detection which *can* be present in some correct tracklet will be present in *some* correct tracklet), but very low tracklet quality (all tracklets are "short" and thus most are low-quality) and mediocre set size.

6.2 Simulated Source Data

The existing MOPS infrastructure includes a powerful simulation layer for generating test data. At its heart is a virtual catalogue of simulated solar system objects with parameters and quantities based on known objects [Grav et al., 2007]. We generate virtual image catalogues using a set of virtual telescope parameters and image observation schedules. Using proper ephemeris calculation, these objects are projected to their expected locations in the sky at the image times, with magnitude a function of object albedo and virtual image exposure time. Optionally, observational error in location and magnitude (assumed to be a Gaussian distribution around the calculated expected location and magnitude) is added. Those point detections which fall within the simulated image radius and the limiting magnitude of the virtual telescope are reported. Optionally, noise observations are distributed throughout the image with a known number of noise points per image, with magnitudes randomly sampled from

a probability distribution which is heavily weighted towards the upper magnitude limits of the virtual telescope.

For simulated LSST data, we will always include observational error, all images are assumed to have exposure time 32 seconds, with limiting magnitude of 25. All images are circular and 9.6 square degrees in area (roughly 1.74 degrees in radius).

Measurements All experiments were carried out on a 4-core AMD Opteron processor with a clock speed 2.194 GHz and 16 GB of RAM. The machine had no other significant load except for basic operating system tasks. All timing information was reported by GNU time.

6.3 Test Runs on A 14-image Stack

We ran exhaustive MHT, the collapseTracklets/purifyTracklets tools described in this thesis, and a simple two-detection tracklet linker on a stack of 14 simulated images from the ecliptic. The stack was large enough to demonstrate the performance problems of exhaustive MHT, but small enough that exhaustive MHT could run without completely exhausting the RAM of the test machine.

Results are presented in 6.3.

6.4 A Very Deep Stack of Simulated LSST Data

6.4.1 Source Data

We tested our tools with a set of detections from the telescope simulation layer. 78 virtual images were taken of the same area of sky along the ecliptic. No noise was added. The time from first to last image was .041852 days, or roughly one hour. No noise was added.

The input data contained a total of 148226 detections of 9156 unique objects.

6.1 shows a graph of the data projected from three dimensions to two.

6.4.2 Evaluation of Results

The following chart (6.4.2) shows our results in terms of accuracy and efficiency. In the left column, we specify the "scoring range": we consider only tracklets and objects whose number of associated detections falls within this range for the subsequent accuracy and efficiency methods on the right.

Method	Run Time	Tracklet Set Size	Object Coverage	Tracklet Quality
simple	2.46	123149	1.0000	0.1706
MHT	248.85	19026124	1.0000	0.5000
MHT-S	225.00	3637	1.0000	0.9445
MHT-SL	225.08	3485	1.0000	0.9842
MHT-SL3+	225.30	2389	0.9086	0.9833
CT	25.66	14646	0.8864	0.3778
CT-S	25.88	7794	0.8856	0.4448
CT-SL	25.96	4373	0.8827	0.6549
CT-SL3+	26.15	2787	0.8191	0.7942

Performance metrics for various methods of tracklet generation. All run times are presented in seconds. Methods tested were as follows:
simple: all feasible two-detection tracklets. Note that run time is short and object coverage is total, but the total output set size is very large and tracklet quality is quite low.
MHT: exhaustive multiple hypothesis enumeration. Run time is high, and tracklet set size is huge. However, the object coverage is total, just as with **simple**. Because longer tracklets are present, tracklet quality is higher than **simple**.
MHT-S: exhaustive MHT with subsets removed. Object coverage, tracklet set size and tracklet quality are high, but run time is unacceptable. Notably, runtime is faster than MHT without subset removal, as the implementation used can perform some clever subset removal in a fairly local manner.
MHT-SL: MHT-S, with the "longest per detection only" post-filter.
MHT-SL3+: MHT-SL, with only tracklets of size 3 or higher considered. Note that this is the only MHT method with less-than-total object coverage; this is because some objects are seen only twice.
CT: collapseTracklets and purifyTracklets.
CT-S: collapseTracklets and purifyTracklets, subsets removed.
CT-SL: collapseTracklets and purifyTracklets, subsets removed and "longest per detection only" post-filter.
CT-SL3+: collapseTracklets and purifyTracklets, same as above, but only tracklets of size 3 or higher are kept. Note that this provides a very significant improvement over CT-SL in all areas but object coverage, where the hit taken is fairly small.

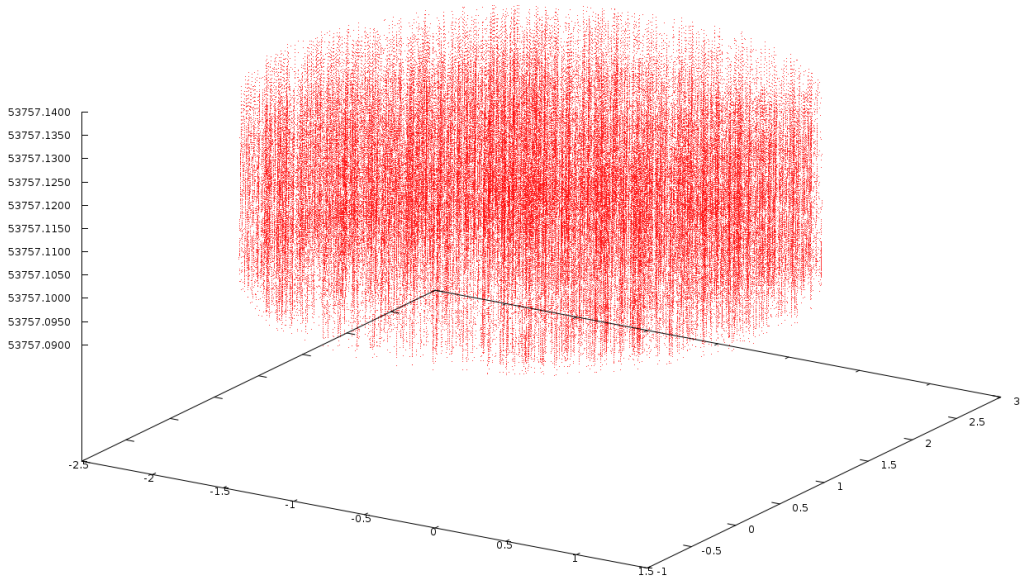

FIGURE 6.1. The test data used for a deep stack. Each point is a detection; the bottom axes are right ascension and declination, the vertical axis is time.

Method	Run Time	Tracklet Set Size	Object Coverage	Tracklet Quality
simple	74.87	4394548	1.0000	0.0316
CT	1729.82	49304	0.8324	0.1809
CT-S	1732.62	33018	0.8300	0.1931
CT-SL	1734.02	12884	0.8280	0.3578
CT-SL3+	1735.69	10575	0.8210	0.4166

Performance metrics for various methods of tracklet generation performed on data from a "deep stack" of 78 images. Labels and units are the same as in 6.3. Note that due to its high memory requirements, exhaustive MHT could not be performed on this data - an object observed 78 times would generate 2^{78} tracklets, more than exhausting the 64-bit address space on our hardware!

Chapter 7

Conclusions

In this thesis, we have presented a methods for deep stack tracklet generation within the MOPS framework as well as metrics for measuring our success.

7.1 Contributions

New methods for deep stack searching and their implementations, as well as additional post-filter implementations. Investigations into the applicability of these heuristics and algorithms for the purposes of moving object search with LSST.

Appendix A

KD-Trees

A.1 Introduction

KD-trees[Bentley, 1975] are a spatial data structure which are used by existing solar system object search methods[Kubica, 2005] as well as the basis for several of the methods presented in this thesis.

KD-trees are a spatial data structure which can be built in $O(N \log N)$ time and allow range searches of the data in $O(\log N)$ time, where N is the total number of data points stored in the tree.

A.1.1 Definition

A KD-tree is a tree structure which holds data using its spatial location as a key. They are generally (and throughout the remainder of this thesis) binary trees: each node holds a pointer to at most two children.

A KD-tree node is either a tree node or a leaf node. A leaf node will hold a data point and has no children. A tree node holds no data, but has pointers to one or two children.

The important detail for KD-trees is that each tree node represents a *partition* of space along one known axis. The partition is *balanced* so that the number of data points held in the "left" partition is equal to the number of data points held in the "right" partition (plus or minus one, in the event of an odd number of total points). All data in the "left" partition is held by the "left" subtree and all data in the "right" partition is held by the "right" subtree. Further, all nodes on the same level of the tree partition data along the same axis, and the axes partitioned at each level are chosen such that each space is partitioned an equal number of times (plus or minus one).

A.1.2 Building A KD-Tree

When building a KD-tree, we consider the dimensionality of our data: say it is n-dimensional. We find the median value of the data along the first axis: call it m. We note this value, store it in the current root node, and create two new tree nodes,

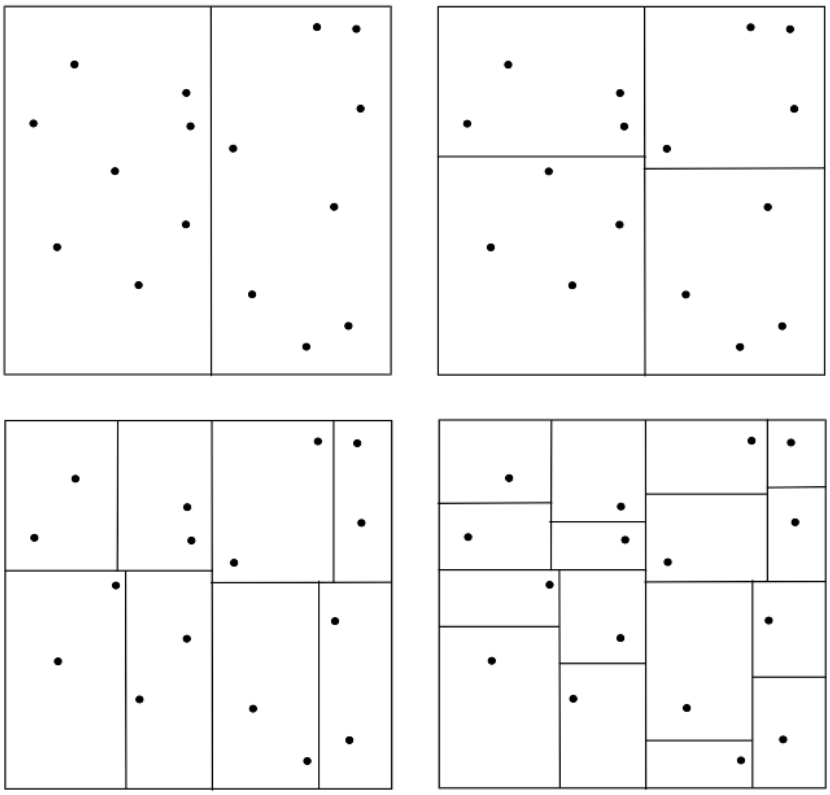

FIGURE A.1. Illustration of recursive partitioning of a 2D space.

and assign all points with value less than m in the first dimension to the "left" node, and all others to the "right" node.

We recursively build each node in a similar way, partitioning along the second axis, then the third, etc., until we hit the nth axis. After this we, partition along the first axis again.

This continues until we create nodes assigned to hold a single data point; these nodes are leaf nodes and hold only one point each.

A.1.3 KD-Tree Range Search

Once a KD-tree has been built, one can perform an efficient range search over the data. Here, a range search is a spatial search over the data, returning all points within a certain radius of a given point.

KD-trees allow us to discard large areas of search space by comparing the area for which a subtree is responsible. In general, range searches take $O(\log n)$ time where n is the nubmer of points in the tree. For an in-depth analysis, see Bentley [1975].

References

J.L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/361002.361007.

K. H. Cook, P. A. Pinto, F. Delgado, M. Miller, C. Petry, A. Saha, P. A. Gee, J. A. Tyson, Z. Ivezic, L. Jones, and LSST Collaboration. LSST: Cadence Design and Simulation. In *American Astronomical Society Meeting Abstracts*, volume 213 of *American Astronomical Society Meeting Abstracts*, pages 460.04–+, January 2009.

O. E. Drummond, W. D. Blair, G. C. Brown, T. L. Ogle, Y. Bar-Shalom, R. L. Cooperman, and W. H. Barker. Performance assessment and comparison of various tracklet methods for maneuvering targets. In I. Kadar, editor, *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 5096 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 514–539, August 2003. doi: 10.1117/12.503750. URL `http://dx.doi.org/10.1117/12.503750`.

Mikael Granvik. *Asteroid Identification Using Statistical Orbital Inversion Methods*. PhD thesis, University of Helsinki, Helsinki, Finland, December 2007.

T. Grav, R. Jedicke, L. Denneau, M. J. Holman, T. Spahr, and Pan-STARRS Moving Object Processing System Team. The Pan-STARRS Synthetic Solar System Model and its Applications. In *Bulletin of the American Astronomical Society*, volume 38 of *Bulletin of the American Astronomical Society*, pages 807–+, December 2007.

Z. Ivezic, J. A. Tyson, R. Allsman, J. Andrew, R. Angel, and for the LSST Collaboration. Lsst: from science drivers to reference design and anticipated data products, 2008. URL `http://www.citebase.org/abstract?id=oai:arXiv.org:0805.2366`.

R. Jedicke. private communication, 2008.

N. Kaiser. Pan-STARRS: a wide-field optical survey telescope array. In J. M. Oschmann, Jr., editor, *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 5489 of *Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*, pages 11–22, October 2004. doi: 10.1117/12.552472.

Jeremy Kubica. *Efficient Discovery of Spatial Associations and Structure with Application to Asteroid Tracking*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, December 2005.

A Milani, G. Gronchi, A. Boattini, N. Kaiser, and Jedicke R. "warning: Sour spots". Technical report, Pan-STARRS, Institute for Astronomy, University of Hawai'i at Manoa, 2680 Woodlawn Drive, Honolulu, Hawaii 96822, March 2006.