Towards Efficient and Precise Queries Over Ten Million Asteroid Trajectory Models

Yusra AlSayyad, K. Simon Krughoff, Bill Howe, Andrew J. Connolly,
Magdalena Balazinska, Lynne Jones

University of Washington, Seattle, WA

Abstract. We consider the problem of finding known asteroids in a given region of space and time. Each asteroid's trajectory is known precisely at any given point in time, but evaluation of the model describing its location is prohibitively expensive at query time. We propose a framework for sampling the base model and indexing the resulting "observations." Our system includes 11 million simulated asteroids, and our requirements include 5 milliarcsecond accuracy and 15 second query response time. We evaluate the effect of various sampling and indexing alternatives in this approach and propose two exemplar solutions: a faster but space-intensive interpolation-based method, or a slower method that relies on bounding regions tailored to the object's behavior but requires fewer sampled points. We implement all solutions in a relational database and evaluate them on a full-scale dataset of known asteroid trajectories.

1 Introduction

The new generation of telescopes under construction return to the same area of the sky with sufficient frequency to allow moving objects — asteroids, near-earth objects (NEOs), comets, and others — to be tracked through direct observation [14, 12]. Studies of asteroids in our Solar System provide a better understanding of the processes that lead to planet formation and enable the identification of potential Earth-impacting asteroids. To detect these moving sources, one image may be subtracted from another (separated in time by several days or weeks) to differentiate variable and moving sources from the dense background of stars and galaxies. Each detected moving source is then compared with a database of expected positions of known asteroids for identification. Linking new observations with known asteroid tracks in this manner is, however, a complex and computationally intensive task subject to strict performance constraints. At a high-level, this task maps onto executing the type of query: "Return all known moving objects that are expected to be located within a given space at a given time." In our context, this problem is challenging for the following reasons:

(1) *Number of Moving Objects (a.k.a. Sources)* Table 1 shows a characteristic Solar System model with over 11 million asteroids [9], including their average and maximum daily motion (in deg. per day). This model represents the types and number of asteroids potentially observable by the Large Synoptic Survey Telescope (LSST) and PAN-STARRS. It is currently being used to simulate the

positions of asteroids within large simulated image streams in order to test the sensitivity of linking and tracking algorithms.

(2) *Complex Trajectory Models* The position of an asteroid can be calculated directly by integrating their tracks using a set of orbital elements[1], accounting for the gravitational interaction with other massive bodies in the Solar System, principally Jupiter. This brute force integration step is prohibitively expensive: For 10,000 sources, the characterization of positions (ephemerides) one years[2] from the epoch of their orbital elements takes 200s using a typical code. To characterize the positions of all ∼11 million sources for all 1000 observations that the LSST will undertake each night would require ∼60 CPU hours.

(3) *Performance Constraints* Over a period of ten years, the Sloan Digital Sky Survey imaged 8,000 sq. deg. (1/5th of the sky), detected $\sim 10^8$ stars and galaxies and amassed approximately 80TB of imaging data. The data rates associated with a new generation of *temporal* surveys, e.g. the LSST, will be three orders of magnitude larger. The LSST requires that we identify all moving sources within the region of the sky encompassed by a single LSST image (i.e. 9.6 sq. deg. circle on the sky) within the 15s required to complete a single exposure of the LSST camera. In short, we have ∼12 hours during the day to build an index (characterize the asteroids' positions for the night), but only 15s to use the index to identify all moving objects found in a single exposure. In one exposure pointed towards the ecliptic, the plane of the Solar System where most asteroids reside, we expect 20k observable main belt asteroids (MBAs).

(4) *Accuracy Constraints* Applications of the asteroid trajectory models typically require high accuracy, precluding the use of coarse statistical approximation methods. The two applications we consider (source identification and source simulation in the LSST simulated image stream) require accuracy to within 1 arcsecond and 5 milliarcseconds, respectively. For comparison, 1 arcsecond corresponds to the apparent size of a dime about 3.7 kilometers away.

Table 1. The 10^7 modeled solar system objects to be queried.

	Daily Displacement	
Count Type	Mean	Maximum
9600k Main Belt Asteroids (MBAs)	0.28°	0.71°
340k Trojans	Varies	Varies
270k Near-Earth Objects (NEOs)	0.5°	150.°
47k Trans-Neptune Objects	0.17°	0.05°
25k Comets	0.16°	32.°
11k Scattered Disk Objects	0.012°	0.05°

Compared to prior work (See Section 4), our problem is thus novel in that we must provide fast look-up queries over objects whose trajectories are described

[1] a set of six variables that completely describe an orbit

[2] Running time scales linearly with the time elapsed from orbital elements' epoch.

by complex models that cannot simply be approximated because of application precision requirements. Within this context, we develop and evaluate two (non-mutually exclusive) approaches for predicting positions of known or simulated asteroids given an arbitrary search area in space and time: (1) modeling each trajectory with a bounding region, using a spatial index to reduce the search space, and evaluating the exact positions of the asteroids at the given epoch using the ephemeris calculation code and (2) modeling each trajectory by a set of positions sampled frequently enough to interpolate their positions within a given accuracy threshold at runtime. We apply these methods to simulations of the data flow from the LSST. In this paper, our goal is to develop a sufficient method to query simulated catalogs for objects that would appear in the LSST's circular aperture at a given pointing and epoch: query time requirement is 30s, storage available is 10TB and accuracy required is 5 milliarcseconds.

The challenge of predicting the positions of sources extends beyond the question of tracking asteroids. In general, our techniques are applicable to a class of spatio-temporal trajectory search problems, where the true positions of the objects can be predicted by the evaluations of complex, often non-linear models that are extremely accurate but computationally expensive. We explore the space, time, and accuracy tradeoffs afforded by pre-sampling the models to offset the cost of evaluating them, but at the expense of storing and indexing these positions coupled with reduced accuracy. Specifically, we contribute

- a description of a challenge problem in trajectory indexing with specific performance, scale, and accuracy constraints, characterized by the availability of precise models of the trajectories (Section 2)
- two methods for solving the challenge problem (Section 2)
- an implementation of each method using a relational database equipped with a spatial index (Section 2),
- an experimental evaluation of the design choices for each technique, and a characterization of the tradeoffs involved (Section 3).

2 Methods

Any realization of the statistical solar system model produces $\sim 10^7$ MBAs. To find all objects in a given region at a given time, the exact positions must be calculated by the predictive model. A naive approach entails calculating the position (ephemeris) of every object in the catalog and comparing that position to the boundary of the search region. Calculating an object's position at epoch t requires expensive numerical integration from the epoch (start time) of the orbital elements, however, sampling the trajectories and storing the sampled positions can significantly reduce the search space. Method 1 samples the trajectories at coarse intervals to reduce the number of possible candidates that must be processed using the expensive ephemeris calculation. Method 2 samples the trajectories at a finer resolution in order to avoid the expensive ephemeris generation step altogether, in favor of a simpler interpolation scheme.

We evaluate these methods to explore the tradeoffs between storage space, query latency, and accuracy, subject to our constraints described in Section 1.

2.1 Method 1

The general outline for method 1 is as follows: 1) *Sample* orbit positions on an intermediate grid (e.g. 1 day), 2) *Group* positions into segments containing two endpoints, then 3) for each segment, *Extrude* a minimal buffer region such that the object does not leave the buffer in the time span bounded by the segment end points. 4) *Index* the bounding regions to support efficient access. 5) *Query* for regions overlapping the search region in space and time to produce a candidate list. 6) *Return* exact positions for each object in the candidate list using the ephemeris generation code. This method is parameterized by the sampling interval, allowing control of the space-time tradeoff: fewer cached positions reduce the database size, but result in a larger list of candidates.

Implementation By joining a table containing one time and position per row (the ephermides table) on itself, each row then contains a start and end position at a start and end time. For each row which represents a 24 hour range of positions, we then created a bounding envelope using Microsoft Spatial.

The naive approach to create a bounding rectangle does not fully contain asteroid trajectories. Figure 1 demonstrates that a trajectory of an asteroid can spend most of its time *outside* a rectangular box of great arc segments. Increasing the size of the rectangles would produce more undesirable false positives.

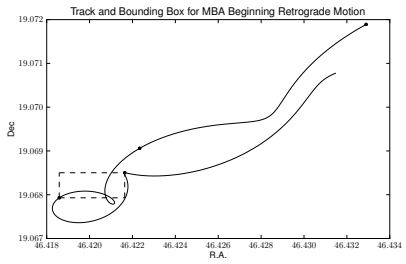

Fig. 1. A rectangular bounding box does not contain the trajectory between the two points.

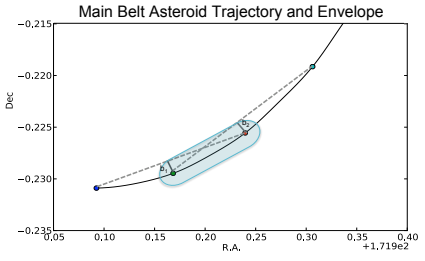

Fig. 2. An ideal bounding box completely encloses the entire track between samples with minimum size.

The nature of the trajectories leads to envelopes described by a line segment with a variable buffer. For example, a nearly linear trajectory should have a small buffer, whereas a curving or looping a trajectory should have a larger buffer to contain the complex motion. The buffer-size is calculated from the velocity the curvature of a trajectory.

To geometrically calculate the buffer $B_{1,2}$ for the line segment (x_1, x_2), we look at how much the trajectory bends entering and leaving the segment. The

bend amount b_i is defined as the distance between point x_i and the great-arc segment (x_{i-1},x_{i+1}). Figure 2 illustrates these definitions.

The envelope buffersize $B_{1,2} = \text{MAX}(b_1,b_2,b_{min})$, where $b_{min} = 400m{=}13arcsec$[3] is a minimum buffer length set by the topographical reflex of the observatory. As the Earth rotates, the asteroids' positions will oscillate. Because period is exactly 1 day, the same as our sampling rate, this motion is imperceptible in the sampled positions and sets an absolute minimum buffer.[4]

2.2 Method 2

The general outline for method 2 is as follows: 1) *Sample* orbit positions on a *fine* grid (e.g., 1 hour), then 2) *Group* enough positions together to parameterize an interpolation method (e.g., four points for a cubic interpolation), then 3) *Index* these positions, 4) *Query* the index to find candidate points within the search region, and finally 5) *Interpolate* to find an approximate position.

This method is faster, but at the expense of data volume and astrometric accuracy: the interpolated positions will always deviate from the positions generated with the ephemeris code by some δ. This method is tunable in that the sampling rate for your fine grid can be chosen such that interpolations deviate by a δ less than your required accuracy.

Implementation To evaluate this method, we sampled the ephemerides every minute for two sets of 1000 randomly selected MBAs. We structured a position table to contain 4 positions and 4 epochs per row, each row corresponding to a one minute range in time and space. Two of the positions anchor the one minute range, and the other two extend one minute beyond on each side to enable cubic interpolation. We store the end points redundantly to increase query speed and accuracy, but note that each row will be shorter than in Method 1 because there is no 144 byte envelope to manage.

In Section 3, we investigate what sampling rate will provide at least a 5 milliarcsecond astrometric accuracy when compared with the model. There are a number of interpolation algorithms to choose from. Cubic spline interpolation is popular and Chebychev polynomials have been used in the astronomy community [15]. To demonstrate our proof of concept we started with a basic third degree polynomial interpolation[17], implemented as a UDF in C#.

3 Evaluation

We implemented both methods for a system that populates simulated LSST images with model MBAs, investigating the challenges and trade-offs of each.

[3] chosen empirically, by observing the maximum reflex.

[4] Another challenge is that MSSQL stores rounded edges as a series of line segments. To reduce data volume, we increased the buffer tolerance, effectively defining an envelope using fewer line segments. The benefit is that the envelopes take 144 bytes on average (instead of default 1232kb), at the cost of storing larger envelopes.

Experimental Setup We start with one instance of the simulated solar system [9] containing 9,655,441 MBAs and their orbital elements as of MJD of 49353.16. We created daily ephemerides for the month starting 49353.16. The production system will ultimately hold 10 years of ephemerides. We loaded the ephemerides into SQL Server 2008, running on a 64 bit system with Windows Server 2008 R2 Standad, 2 Quad-Core Xeons @ 2.26 Ghz, 24GB RAM. The RAID controller was a Dell PERC 6/E with 512MB buffer supporting for SAS.

Method 1 Each row contains a start and end time, a start and end position, and a bounding envelope calculated as described Section 2. We indexed these envelopes using the default MS Spatial Index and ran performance tests for queries pointing at regions of different densities: the ecliptic (most dense), 10°, and 20° away from the ecliptic. These density bins were chosen in effort to separate the query time's dependance on the size of the search area and the number of envelopes returned. At each pointing, we increased the circular search area's radius 2X and recorded the query time. Figure 3 shows the query time curves for the three densities, plotted both by search area and envelopes returned to demonstrate the dependence on each. For search areas greater than 10 sq. deg. the query time scales linearly with the bounding boxes returned independent of search area, possibly because the MS Spatial intersection algorithm behaves differently when the search area increases beyond the size of the envelopes. We can return envelopes which intersect a 9.6sq. deg. search area in ∼20s, when pointed towards the densest area of the sky.

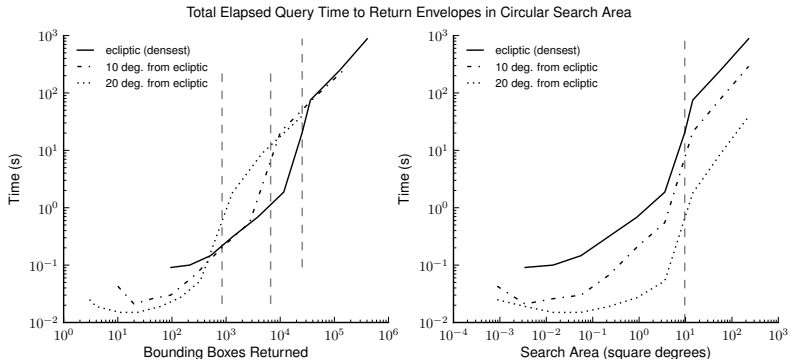

Fig. 3. Performance using MS Spatial Index for three search regions and densities. The density decreases as the search area deviates from the ecliptic. Vertical lines represent search area of 9.6 sq. deg. The query times are plotted against envelopes returned (left) and against search area (right).

From here, the system returns the orbital elements of these asteroids for calculation of their exact positions using the ephemeris code, OpenOrb [3]. We found that ephemeris calculation time increases linearly with time since

the orbit element's epoch. For 10,000 objects on a 2.66 GHz machine: $t \approx (days\ from\ orbital\ epoch) \times 0.55(s/day) + 3.5s$

The ephemeris calculation time can be mitigated by storing the orbital elements for every year, month or week to balance storage and query time requirements. An orbital element table for 10^7 asteroids fills 2GB. Because ephemeris calculation can run forwards *or backwards*, if orbital elements were stored every 2 weeks, the longest an orbit would need to be propagated is 7 days. This corresponds to a runtime of 15s for the "worst case" ecliptic search. The final step compares each position with the search area boundary, which takes < 1s.

In this method the resulting accuracy is *exact* with respect to our model. Storage volume and query time can be balanced depending on the available space and query time requirements. For our available storage space a "worst case" query would take 20s to return the bounding boxes and 15s to to calculate the ephemerides. For one month of data, the daily ephemerides (460M rows) fill approximately 50GB and two sets of orbit elements 4GB. For context, our system requires functionality over 120 months.

Method 2 To determine the minimum time, dt, between sampled positions to provide 5 milliarcsecond accuracy, we created sets of interpolated positions from $dt = \{(2, 4, 8, 32, 64, 128, 256, 512\}$ minutes and compared the interpolated positions with the model-produced positions to create error sets. Figure 4 shows the distribution of errors for 3 different sampling rates. A factor of 2 increase in dt causes an order of magnitude increase in the interpolation error. Fast-moving asteroids and complicated-moving asteroid require most frequent sampling. However, we found that fast movers have linear tracks and those with complicated tracks move slowly, which narrows the necessary sampling rate.

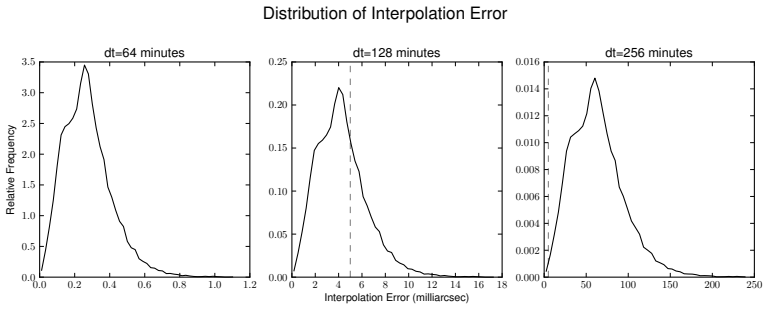

Fig. 4. Distribution of interpolation errors for three constant sampling rates: $dt = \{64, 128, 256\}$. Dashed vertical lines mark the $5mas$ accuracy requirement.

We find that we need $dt = 64$ to achieve 5 mas accuracy. Using an adaptive refinement algorithm which stores only necessary samples, we can achieve this accuracy with a combination of $dt = 64$ and $dt = 128$. This would require approximately 16 positions to be stored for 1 day or 7.4B rows per month.

Next, we investigate the query times. Because the interpolation is fast (<1s for 40k MBAs), we do not need such strict performance from the coarse spatial filter. Therefore, we query by extending the search area by the maximum displacement by an MBA in dt (this only works with a constant dt), treating positions as points rather than polygons. We use a simple hierarchical triangular mesh (HTM) index [7] to return the points in the set of triangular pixels covering the search area. Next, we find the interpolated positions, and finally compare the interpolated positions with the circular search area. Querying against the HTM index and comparing the interpolated positions combined takes ∼300ms for 40k objects, achieving our required total query time on the order of seconds.

4 Related Work

Moving objects databases are a well established research area [10], although commercial DBMSs provide only support for spatial data types [1, 2, 4].

Most closely related work studies techniques for indexing moving object trajectories. Traditional techniques commonly assume that objects have linear trajectories in one or two dimensions [5, 8, 13]. In our applications, moving objects follow complex paths. Recent work [6] studied the problem of indexing objects with non-linear trajectories in arbitrary dimensions by indexing the parameterized representation of these trajectories. This is an approach orthogonal to the ones that we study in this paper.

In general, our problem is unique in that it combines the requirement for (1) handling an extremely large number of objects whose (2) trajectories are described with complex models, and (3) answering range-selection queries on these objects wthin strict time and precision constraints. Within this context, we studied how to uniquely combine various existing techniques to derive an efficient solution. The related techniques include trajectory sampling and position interpolation [10], creating buffers around trajectories to capture uncertainty in object location [10], splitting a trajectory into minimum bounding regions shaped like hyperrectangles [11, 18] or more complex shapes [19], and polynomial trajectory approximation [16].

5 Conclusion and Future Work

We tested two methods for storing and querying asteroid positions based on their position at a given time. The bounding envelope method has the benefits of exact accuracy and tunable storage requirements, but at the cost of lengthy query times. By testing the interpolation method we found that we can reduce the query time drastically, but at the expense of either accuracy or storage space. In the evaluation, we held the accuracy requirement fixed to investigate the required storage. Using a third degree polynomial interpolation, we would exceed our available storage (10 years of data would fill ∼900B rows).

We are currently investigating how the interpolation (cubic spline, Chebychev polynomials) changes the accuracy, and indirectly the data volume. Furthermore, in effort to reduce the query time in method 1, we are tweaking the balance between the sampling interval of bounding envelopes and how frequently to store the orbital elements, the input for ephemeris generation. Ultimately, the storage/query time/accuracy parameter space would be fully explored providing an optimal solution for an arbitrary set of project requirements.

Acknowledgments This work is funded by the National Science Foundation Cluster Exploratory (CluE) grant (IIS-0844580).

References

1. IBM spatial offerings. `http://www-01.ibm.com/software/data/spatial/`.
2. Microsoft SQL Server 2008: Spatial data. `http://www.microsoft.com/sqlserver/2008/en/us/spatial-data.aspx`.
3. Openorb. `http://code.google.com/p/oorb/wiki/OpenOrb`.
4. Oracle spatial user's guide and reference (release 9.0.1). `http://download.oracle.com/docs/html/A88805_01/sdo_intr.htm`.
5. P. K. Agarwal, L. Arge, and J. Erickson. Indexing moving points. *J. Comput. Syst. Sci.*, 66:207–243, February 2003.
6. C. C. Aggarwal and D. Agrawal. On nearest neighbor indexing of nonlinear trajectories. In *Proc. of the 22nd ACM Symp. on Principles of Database Systems (PODS)*, pages 252–259, 2003.
7. T. Budavari, A. Szalay, and G. Fekete. Searchable Sky Coverage of Astronomical Observations: Footprints and Exposures. *PASP*, 122:1375–1388, Sept 2010.
8. H. D. Chon, D. Agrawal, and A. E. Abbadi. Storage and retrieval of moving objects. In *Proceedings of the Second International Conference on Mobile Data Management*, MDM '01, pages 173–184, 2001.
9. 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. volume 38 of *Bulletin of the American Astronomical Society*, pages 807–+, Dec. 2007.
10. R. H. Güting and M. Schneider. *Moving Objects Databases*. Morgan Kaufmann, 2005.
11. M. Hadjieleftheriou, G. Kollios, J. Tsotras, and D. Gunopulos. Indexing spatiotemporal archives. *VLDB Journal*, 15:143–164, June 2006.
12. N. Kaiser and et al. Pan-STARRS: A Large Synoptic Survey Telescope Array. volume 4836 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*, pages 154–164, Dec. 2002.
13. G. Kollios, D. Gunopulos, and V. J. Tsotras. On indexing mobile objects. In *Proc. of the 18th ACM Symp. on Principles of Database Systems (PODS)*, pages 261–272, 1999.
14. LSST Science Collaborations. LSST Science Book, V2.0. *ArXiv*, Dec. 2009.
15. X. X. Newhall. Numerical Representation of Planetary Ephemerides. *Celestial Mechanics*, 45:305–+, 1989.
16. J. Ni and C. V. Ravishankar. Indexing spatio-temporal trajectories with efficient polynomial approximations. *IEEE TKDE*, 19:663–678, May 2007.

17. W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical recipes in C (2nd ed.): the art of scientific computing*. Cambridge University Press, New York, NY, USA, 1992.
18. S. Rasetic, J. Sander, J. Elding, and M. A. Nascimento. A trajectory splitting model for efficient spatio-temporal indexing. In *Proc. of the 31st VLDB Conf.*, pages 934–945, 2005.
19. H. Zhu, J. Su, and O. H. Ibarra. Trajectory queries and octagons in moving object databases. In *Proceedings of the eleventh international conference on Information and knowledge management*, CIKM '02, pages 413–421, 2002.