Online Amnesic Summarization of Streaming Locations

Michalis Potamias[1], Kostas Patroumpas[2], and Timos Sellis[2]

[1] Computer Science Department, Boston University, MA, USA
[2] School of Electrical and Computer Engineering
National Technical University of Athens, Hellas
`mp@cs.bu.edu, {kpatro, timos}@dbnet.ece.ntua.gr`

Abstract. Massive data streams of positional updates become increasingly difficult to manage under limited memory resources, especially in terms of providing near real-time response to multiple continuous queries. In this paper, we consider online maintenance for spatiotemporal summaries of streaming positions in an aging-aware fashion, by gradually evicting older observations in favor of greater precision for the most recent portions of movement. Although several *amnesic* functions have been proposed for approximation of time series, we opt for a simple, yet quite efficient scheme that achieves contiguity along all retained stream pieces. To this end, we adapt an amnesic tree structure that effectively meets the requirements of time-decaying approximation while taking advantage of the succession inherent in positional updates. We further exemplify the significance of this scheme in two important cases: the first one refers to trajectory compression of individual objects; the other offers estimated aggregates of moving object locations across time. Both techniques are validated with comprehensive experiments, confirming their suitability in maintaining online concise synopses for moving objects.

1 Introduction

The tremendous amount of information flowing as transient *data streams* in many modern applications that monitor the current position of people, vehicles, animals etc. or track their trajectories, clearly calls for single-pass processing. It is inevitable that not all data can be retained permanently in memory, so stale items can either be archived on disk or even discarded to make room for newly arriving tuples. Sometimes, it is indispensable to maintain a finite *window* over the stream, which provides access to the most recent stream items [1].

More importantly, the significance of each isolated positional tuple is *time-decaying*: when it first arrives, it perhaps conveys critical information, but it gets less and less important as time goes by, until it eventually becomes obsolete and practically useless. Therefore, the older a data item gets, the coarser its representation could become in a progressive fashion, implying that greater precision should be reserved for the most recent items. This is essentially reminiscent of the way human memory actually works: we can accurately describe recent events in much detail, but we can hardly recollect facts that occurred a long time ago.

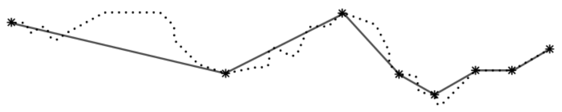

Fig. 1. Amnesic approximation of a trajectory.

This treatment of data has been termed *amnesic* with respect to time series approximation [9], in the sense that acceptable error margin in data is allowed to increase in proportion to its age. A wide range of amnesic functions was identified for controlling the amount of error tolerated at every single point in the time series. Piecewise linear approximation (PLA) functions were superior in terms of incremental computation and availability of various distance metrics [9].

In this paper, we turn our focus on amnesic approximation of spatiotemporal data streams generated by tracking a large number of moving objects. This approach is mainly dictated by the sheer volume of data: trajectories can become rather lengthy when continuously accumulating positional updates, and thus difficult to accommodate in full precision for their entire history. In addition, different levels of abstraction are also inherent in semantics related to multi-scale representation of spatial features, given that more precision is allocated to a user-defined area of interest. In a spatiotemporal context, this specific interest can be interpreted as the current or most recent location of objects.

Although our primary interest is on spatiotemporal streams, the framework we propose can certainly be applied over typical streams comprised of items in a sequence, such as sensor readings or stock tickers across time. In particular, we present the *multiple-granularity* AmTree framework, by adapting a tree structure from [3]. AmTree accepts streaming items and maintains summaries over hierarchically organized levels of precision, realizing an amnesic treatment over stream portions. This mechanism can produce reduced data representations for approximate query answering, by efficiently handling streaming locations from moving point objects or retaining a rough outline of their trajectories. As illustrated in Fig. 1, this summary keeps more dense information for the recent past, while older segments are evidently underrepresented. As we show in this paper, not only is this specific framework proven sufficient to cope with online trajectory approximation, but it can further be used in spatiotemporal aggregation, in order to estimate distinct count queries over locations of moving objects.

The distinguishing characteristics of our approach as opposed to general piecewise linear approximation are twofold. First, we preserve *contiguity* among successive trajectory segments for each individual object, no matter how coarse the level of approximation gradually becomes. In fact, we do not purposely introduce fictitious points to obtain a smoother approximation, but we retain a subset of actual locations to keep the reduced representation consistent to the original data. Second, the *hierarchical* behavior of our transform fits well to the streaming nature of incoming locations. Our amnesic tree structure gradually achieves a coarser representation for each trajectory without sacrificing its latest details.

Our contributions can be summarized as follows:

– We propose a generic structure named AmTree and we formalize its basic operations. AmTree exploits the notion of temporal timeliness and is capable to maintain a compact amnesic representation of streaming items.
– We demonstrate how this structure can be applied to address two challenging issues in spatiotemporal data compression, namely concise approximation of entire trajectories and distinct count estimation over moving objects.
– We present an extensive experimental study of our techniques, using large synthetic datasets. These experiments confirm the robustness of the proposed structure and the high-quality synopses it produces.

The remainder of this paper proceeds as follows. Section 2 discusses some preliminary concepts. In Section 3, we present the structure of AmTree along with its basic operations and characteristics. In Section 4, this scheme is utilized to maintain amnesic approximations at a single-trajectory level. In Section 5, we introduce a composite structure based on AmTree capable to estimate distinct counts of numerous moving objects. Experimental results are discussed in Section 6. Section 7 reviews related work, while Section 8 offers concluding remarks.

2 Aging Stream Features at Multiple Time Granularities

Time dimension is intuitively liaised to several levels of detail with respect to a time domain. A *time granule* is a unified set of discrete time instants that can be used as a time reference for data items. Consecutive and non-overlapping time granules can be merged into greater ones in an iterative fashion, thus defining several levels of *granularity* [2], like seconds, minutes, hours, days etc.

More concretely, let $G_0, G_1, \ldots, G_{k-1}$ denote k successive levels of granularity, respectively characterized by a granule unit $\delta_0, \delta_1, \ldots, \delta_{k-1}$. Granule unit δ_i at level i consists of a fixed set of primitive time instants, e.g., an hour granule consists of 3600 seconds (assuming seconds as primitive instants). In our approach, we require that each granule unit δ_i at level i subsumes a fixed number of contiguous non-overlapping granules of unit δ_{i-1} at level $i-1$. Assuming that granularity level G_i consists of n granules $g_0^i, g_1^i, \ldots, g_{n-1}^i$, each such granule is equivalent to a fixed-size finite set of contiguous instants. In general, it may occur that the number of granules making up the immediately higher granule (i.e., the *granularity factor*) varies across levels, as occurs for hours, days, months etc.

The whole process implies a hierarchical composition of granules, as long as their endpoints coincide throughout different levels of granularity. Days, months and years can be defined in that order one from the other, since the intervals they span are entirely contained in a granule higher up in the hierarchy. Apparently, such a hierarchical scheme can be effectively represented by a *tree*: its lower level corresponds to the finer granules, and each successive level higher in the tree to coarser ones, up to the root level that denotes the coarsest granule available. By simply ascending or descending that tree, we achieve varying levels of detail.

When it comes to data representation, a hierarchical scheme can be advantageous in referencing aging stream portions. A *data stream* S may be regarded as an ordered sequence of items $\langle s, \tau \rangle$, where s is a typical relational tuple (e.g., carrying object id's and positions) and τ its timestamp value.

We envisage a mechanism where streaming tuples are taken in at the finest granularity δ_0; periodically, as soon as all items spanning the duration δ_i of any given level $i > 0$ have been received, they are combined into a summary assigned to a granule at level i. Although this process is similar for all levels, it is applied at time instants that denote endpoints of the respective granules. For instance, stream items may arrive every second and a synopsis is emitted each minute; when 60 such synopses have been produced, a more coarse synopsis is made up over past hour and so on, until the higher level (e.g., year) is reached. In addition, data items of a finer granule unit δ_{i-1} could get discarded as soon as they are summed up at a coarser granule unit δ_i. This way, less and less detailed information will be maintained for older stream portions, practically implementing a deterministic *time-decaying* policy regarding data stream summarization.

3 The Amnesic Tree

Next, we present the proposed AmTree structure, which maintains a time-aware, hierarchical amnesic synopsis of a data stream. As it will become evident soon, we opt for a summarization scheme that manipulates pairs of items at every granularity level. Still, the proposed framework is much more general and can be easily calibrated to work with varying number of granules at each level.

3.1 Structure and Properties

The general structure of an AmTree is illustrated in Fig. 2. We enumerate granularity levels of the tree starting from the lowest one (0^{th} level). We assume that the granularity factor is 2; hence, a granule at each level spans two granules half its size at the level beneath. Except for the root, each level i of the tree consists of two nodes, the right (R_i) and the left one (L_i). At the 0^{th} level, each node accepts data with reference to the finest granularity unit δ_0, which characterizes every timestamp attached to incoming tuples. Each node at the i^{th} level contains information about twice as many timestamps as a node at the $(i-1)^{th}$ level. Hence, a node at level i contains information characterizing 2^i timestamps.

Let N denote the number of stream items received thus far by AmTree, each one carrying information at the finest granularity unit δ_0. Then, we can easily calculate the height H of the tree, i.e., the total number of levels, as $H = \lfloor \log_2 N \rfloor + 1$. Thus, the number of tree nodes is $2 * H - 1$.

Without loss of generality, we assume that AmTree accepts tuples with distinct successive timestamps, so we adhere to a count-based stream model where each new item is timestamped with the next available sequential integer value. Even without this precondition, we may assume a mapping f that is applied over the batch of tuples with current timestamp value τ and transforms them into a single tuple that can be the content of a tree node. As shown in Fig. 2, the resulting content is assigned to node R_0, while the previous content of R_0

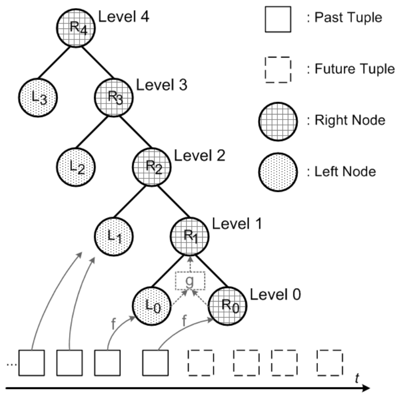

Fig. 2. Basic structure of AmTree.

is shifted to node L_0. As time goes by and new data comes in, the contents of each level are combined using a function g and propagated higher up in the tree, retaining less detail. This process is performed using the following methods:

– $new(\tau)$ is invoked as soon as a new item of timestamp τ has arrived and it simply assigns the result of mapping f into node R_0. Note that node R_0 is the only entry point to the synopsis prepared by the AmTree.
– $shift(i)$ discards the contents of node L_i and shifts contents of R_i into L_i. If node L_i does not exist, then $shift(i)$ allocates appropriate node space.
– $merge(i)$ invokes function g over the contents of L_i and R_i and assigns their combined result to node R_{i+1}. If R_{i+1} does not yet exist, then node space is allocated, effectively increasing the height H of the tree.

3.2 Streaming Operation

In stream processing, *sliding windows* are utilized to fetch the most recent items qualifying to a standard extent of interest in terms of time interval (e.g., ten minutes long) or number of stream tuples (e.g., 100 tuples). *Landmark windows* specify a fixed anchor point in time and they continue appending newly arriving tuples (e.g., get all items after 10 a.m. today). Our intention is to handle stream elements in a way that combines the characteristics of both window modes. While we need to follow stream evolution in a sliding fashion, we must still keep some reference to its long-running history after a specific "landmark" in time. This approach actually resembles a *tilt time frame* model [4] and fulfils the requirements of a time-decaying representation.

We will demonstrate the update procedure with an example depicted in Fig. 3. For clarity, nodes are shown filled with timestamps and not with the actual stream values. We illustrate 9 successive states of AmTree, starting (Fig. 3a) from a snapshot after 16 insertions (i.e., timestamps $0, 1, 2, \ldots, 14, 15$ have been processed). The case when tuple with $\tau = 16$ is inserted into the tree is depicted

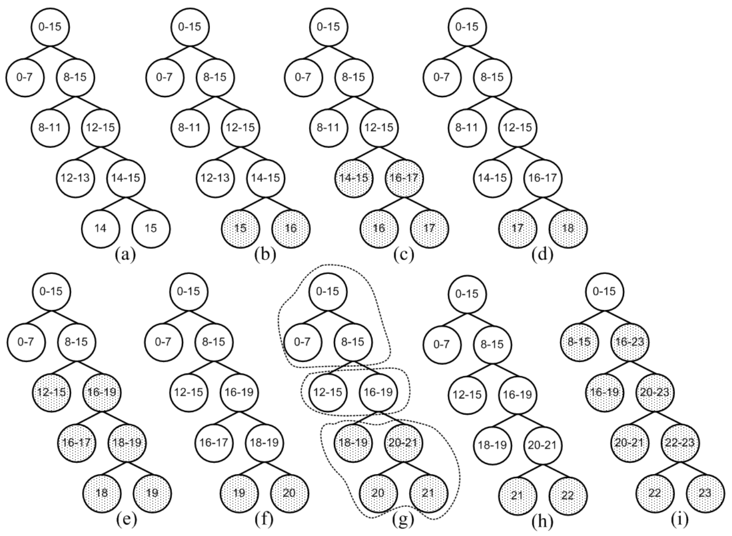

Fig. 3. Successive snapshots of an AmTree.

in Fig. 3b, where all updated nodes are highlighted. In particular, $shift(0)$ is invoked to transfer the current contents of R_0 into L_0, and subsequently $new(16)$ assigns a new content to R_0, as always occurs at every new insertion.

When tuple of $\tau = 17$ arrives, another update call is invoked; this time, level 1 will be also updated as long as both nodes of level 0 have entirely new information since the last update of level 1 occurred (Fig. 3c). Level 0 is processed in exactly the same way as before, while level 1 is processed in two steps: first, $shift(1)$ is employed to discard contents of L_1 and transfer R_1 to L_1; consequently, $merge(0)$ causes the new content of R_1 to be a combination of R_0 and L_0.

Observe that updates work in a bottom-up fashion. So, level 0 will be updated at every insertion, by always calling $shift(0)$; $new(\tau)$. Accordingly, level 1 will be updated every second tuple, because only then contents of level-0 nodes have not yet been merged. So, whenever the update procedure reaches level 1, calls $shift(0)$; $new(\tau)$; $shift(1)$; $merge(0)$ must have been invoked. At $\tau = 19$ (Fig. 3e), levels 0, 1 and 2 will be updated, but updates will not proceed to level 3, since R_3 still temporally overlaps with L_2.

Intuitively, level 2 will be updated every fourth tuple, level 3 every eighth tuple and so on. This example indicates that each level i is updated every 2^i timestamps ($i = 0, 1, 2, \ldots$). We can now predict that level 3 is updated at timestamps 7, 15, 23 (Fig. 3i), and so on. At each incoming timestamp, the update procedure ascends the tree up to a maximum level M, which depends on the N stream items seen so far. Clearly, M can be derived as the power of 2, if N is analyzed in its prime factors, i.e., $M = max\{k : \exists\ \gamma \in \mathbb{N},\ \gamma \cdot 2^k = N\}$.

Updating AmTree is clearly an online technique that incurs logarithmic memory storage and can be performed quite quickly. More formally:

Lemma 1. *The space complexity of the AmTree structure is $O(\log N)$.*

Lemma 2. *The amortized time complexity of the AmTree update procedure is $O(1)$ per stream tuple.*

Proof. Each level i of the AmTree is updated every 2^i timestamps and we assume cost $O(1)$ to update a single level. So, when N tuples (i.e., timestamps) have arrived, every level has been updated $\lfloor \frac{N}{2^i} \rfloor$ times. Since the total number of tree levels is $H = \lfloor \log N \rfloor + 1$, this results to a cost of $\sum_{i=0}^{H-1} O(1) \frac{N}{2^i} = O(N)$, because $\sum_{i=0}^{H-1} \frac{1}{2^i} < 2$. Thus, the amortized update cost per tuple is $O(1)$. □

3.3 Multiple Concurrent AmTrees

The basic AmTree structure (Section 3.1) has non-tunable characteristics that determine its aging approximations. Relaxing this strict control over time-decaying rate, we can generalize it into a forest of AmTrees of varying resolution.

AmTreeH is a set of m *concurrently maintained* AmTree structures. When a stream tuple arrives for processing, it gets inserted into just one AmTree. The appropriate AmTree$^{(p)}$ is determined by a hash function $p = h(\tau)$ computed over the current timestamp value τ. Therefore, amortized update complexity is still $O(1)$ per tuple, while worst-case space complexity is $O(m \log N)$.

AmTreeH achieves tunable amnesic features depending on the mappings performed by function h. To exemplify its usage, we consider hash function $p = h(\tau) = \max\{i \in \mathbb{N} : \exists\, \lambda \in \mathbb{N},\ \tau = 2^i + \lambda \cdot 2^{i+1}\}$, $\tau = 1, 2, 3, ...$, which assigns tuples into any given $AmTree^{(p)}$. Figure 4 illustrates a snapshot of this multi-AmTree structure after tuple with $\tau = 1024$ has been inserted. Assuming tuples of distinct timestamps, items with $\tau = 1, 3, 5, 7, \ldots$ are inserted into $AmTree^{(0)}$, items with $\tau = 2, 6, 10, 14, \ldots$ get inserted into $AmTree^{(1)}$, those with $\tau = 4, 12, 20, 28, \ldots$ are consumed by $AmTree^{(2)}$, etc. A new AmTree is initiated whenever a tuple of timestamp equal to a power of 2 is received. So, another AmTree will be created as soon as tuple of $\tau = 2048$ arrives.

Essentially, this hash function demultiplexes the incoming stream into several substreams consumed by diverse trees. In fact, the first $AmTree^{(0)}$ is updated every second tuple consuming half of the incoming items, the second $AmTree^{(1)}$ is updated every fourth tuple consuming a quarter of the total items and so on. Clearly, $AmTree^{(0)}$ evolves rapidly, $AmTree^{(1)}$ is updated more slowly (at the half rate) and so on. Thus, any given $AmTree^{(p)}$ has one level less compared to its predecessor $AmTree^{(p-1)}$. The overall space complexity of AmTreeH is $O((\log N - 1) + (\log N - 2) + ... + 1) = O((\log N - 1) \cdot \frac{1 + \log N - 1}{2}) = O(\log^2 N)$. Meanwhile, amortized update complexity still remains $O(1)$ per tuple, since each item is inserted into exactly one AmTree.

Not only does this scheme provide more dense approximations, but it may be also valuable for multi-resolution stream representations. Thus, a query may be given a quick approximate answer using data from the last *AmTree* in the forest. Upon user request, further refinements may be retrieved from other trees with more levels, gradually enhancing result with extra details.

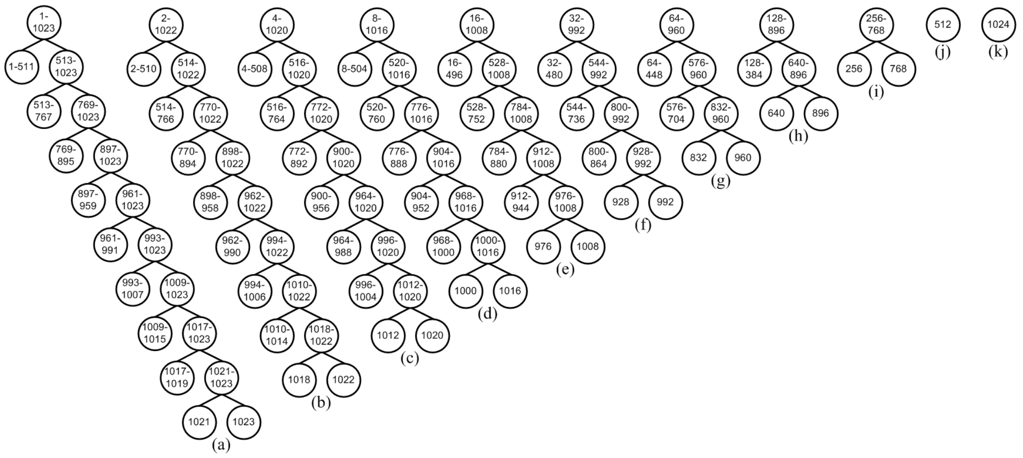

Fig. 4. Multiple concurrent AmTrees.

4 Amnesic Trajectory Synopses

So far, we have presented our generic summarization structure, but we believe that AmTree is best suited for summarizing streams of sequential features, i.e., time series that must retain contiguity among their consecutive elements. This is exactly the case of streaming locations that track the movement of many moving objects across time. In this section, we will study amnesic synopses concerning singleton trajectories, before proceeding to give a technique for computing aggregates over moving objects in Section 5.

4.1 Linear Representation of Trajectories

Consider a set of numerous point objects continuously moving over the Euclidean 2-dimensional plane. Obviously, the evolving sequence of locations recorded for each object constitutes its trajectory. Since it is not realistic to maintain a continuous trace of each object, only point samples can be practically collected from the respective data source at distinct time instants (e.g., every few seconds a car sends its location measured by a GPS device).

Therefore, *trajectory* T of a point object moving over the Euclidean plane is a possibly unbounded sequence of timestamped locations across time. Thus, we consider a sequence of tuples $\langle oid, \tau_i, x_i, y_i \rangle,\ i = 1, \ldots, N, \ldots$, assuming that for an object with identity *oid*, spatial position (x_i, y_i) has been recorded at timestamp τ_i. Note that this representation implies that a trajectory is essentially a collection of points put in order by their timestamp values.

Instead of such a "chain" of points, a linear representation of each trajectory is sometimes more adequate, since it conveys the sense of a continuous, though approximate, trace of its movement. Such a polyline is composed of consecutive line segments, each one connecting a pair of successive point locations recorded for this object. Effectively, each line segment indicates the *displacement* of an object between two successive recordings, that is, it reveals the change of its position with respect to its previously known location. More concretely, let two successive positional updates $\langle oid, \tau_i, x_i, y_i \rangle$, $\langle oid, \tau_j, x_j, y_j \rangle$ of the same moving object *oid*, where $\tau_i < \tau_j$ and $\nexists\ \langle oid, \tau_k, x_k, y_k \rangle, \forall\ \tau_k \in\ (\tau_i..\tau_j)$. Then, we can calculate in constant time the displacements $dx = x_i - x_j$ and $dy = y_i - y_j$ during time interval $[\tau_i..\tau_j]$ along axes x, y respectively. This process can be applied in an incremental fashion for each object; every time a positional update arrives, this calculation requires just the (already maintained) last known location.

Therefore, as an alternative representation to a time series of points, a trajectory can be trivially transformed to a sequence of displacements of the form $\langle oid, \tau_{start}, \tau_{end}, dx, dy \rangle$. In fact, $\tau_{start} = \tau_i$ and $\tau_{end} = \tau_j$ denote the bounds of the corresponding time interval, but, for storage savings, we may even omit attribute τ_{end} from the schema of tuples, thanks to contiguity property.

4.2 Updating Trajectory Synopses

For compressing a single trajectory, we suggest an instantiation of AmTree that manipulates all successive displacement tuples recorded for this object. In direct correspondence to AmTree functionality presented in Section 3.1, mapping f converts each current position $\langle oid, \tau_{cur}, x_{cur}, y_{cur} \rangle$ into a displacement tuple $\langle oid, \tau_{prev}, \tau_{cur}, dx, dy \rangle$. This tuple is then inserted into node R_0, possibly triggering further updates at higher levels as mentioned in Section 3.2.

When the contents of level i must be *merged* to produce a coarser representation, a function g must be invoked in order to combine the displacements stored in nodes L_i and R_i. In the case of trajectories, this can be handled as a *concatenation* of two trajectory segments that span consecutive time intervals of identical size. More formally, let two displacement tuples l_1, l_2 concerning the same object *oid* with $\tau^{(1)}_{end} = \tau^{(2)}_{start}$, that are stored in nodes L_i and R_i, respectively. Function $g(l_1, l_2)$ is used to concatenate them into a single tuple $l \equiv l_1 \circ l_2 = \langle oid, \tau^{(1)}_{start}, \tau^{(2)}_{end}, dx^{(1)} + dx^{(2)}, dy^{(1)} + dy^{(2)} \rangle$. This concatenation inevitably leads to a loss of information, since the common articulation point of segments l_1 and l_2 is removed altogether, i.e., the point location that was recorded at timestamp $\tau^{(1)}_{end} = \tau^{(2)}_{start}$. In effect, the resulting displacement l is defined solely by the two non-common endpoints of l_1, l_2, which are exactly the locations recorded at timestamps $\tau^{(1)}_{start}$ and $\tau^{(2)}_{end}$. This process can be easily generalized to $k \geq 2$ consecutive displacement tuples $l_1, l_2, \ldots l_k$, retaining only the first and the last point location in this subsequence.

It must be stressed that endpoints of all displacements stored in AmTree nodes correspond to original positional updates, while displacements remain connected to each other at every level. However, as trajectory information propagates into higher levels of the tree, line segments of increasing temporal extent and less positional accuracy get created by progressively eliminating intermediate point locations. Thus, when going higher in each tree snapshot, more rough segments are derived for the more distant trajectory parts, while more precise segments at the lower levels correspond to recent portions of its movement. Evidently, an amnesic behavior is realized for trajectory segments through levels of gradually less detail in such bottom-up tree maintenance.

4.3 Reconstructing Trajectories from Synopses

Trajectory synopses accomplished by AmTree can give a reduced representation of an object's movement utilizing several tree traversal schemes. As long as consecutive displacements are preserved, we are able to reconstruct the movement of a particular object starting from its most recent position and going steadily backwards in time by choosing points in descending temporal order. Any trajectory reconstruction process can be gradually refined by combining information from multiple levels and nodes of the tree.

A simple reconstruction scheme would be a tree traversal in a bottom-up fashion, starting from node R_0 and then accessing right and left nodes at each level. Yet, many nodes that do not contribute any additional points will also be visited. In the snapshot at Fig. 3a, none of the L_i nodes will provide any additional point, since their content is already merged into their parent R_{i+1}.

By construction, all R_i nodes contain displacements concerning gradually aging positions of an object, as their time granules span more extended periods in the past. Besides, each new location does not necessarily incur changes to all tree levels, but up to a maximum level M that depends on the arrival order of this point. But how many of those L_i nodes should be used to further refine any trajectory approximation? Further, which is the minimal set of these nodes?

The important observation is that tree updates take place in waves that involve levels up to M. Each such wave synchronizes these levels to the current timestamp value, but nodes at levels higher than M remain completely untouched, meaning that the granules of R_M and R_{M+1} are time intervals that do not end up at the same timestamp value. Thus, after each insertion, consecutive tree levels are clustered into groups according to the time instant they were updated for the last time, as shown in Fig. 3g. Intuitively, if we were able to locate at which level each successive wave has finished its updates, we could get the required L_i nodes as these are the upper left nodes of each such cluster.

Let c denote the number of L_i nodes that are able to contribute to trajectory reconstruction. Not surprisingly, c can be easily computed from the number of 1's in the binary representation of the current number N of positional updates received thus far for this particular object, after excluding the most significant bit (since always MSB=1). Referring back to Fig. 3g, when point with timestamp $\tau = 21$ arrives, it is the 22nd tuple in the sequence; hence, $N = 22$ and its binary representation (10110) contains three 1's, so $c = 2$ after excluding the MSB. It is precisely the position of those 1's in this binary representation that dictates the level of the L_i nodes that suffices to be visited, i.e., left nodes at levels 1 and 2. Hence, for reconstructing the trajectory that corresponds to the snapshot of Fig. 3g, apart from all R_i, nodes L_1 (18-19) and L_2 (12-15) should also be taken.

In total, all H (height of the tree) of the R_i nodes and a number c of the L_i nodes can provide all knowledge contained in any AmTree state. This reconstruc-

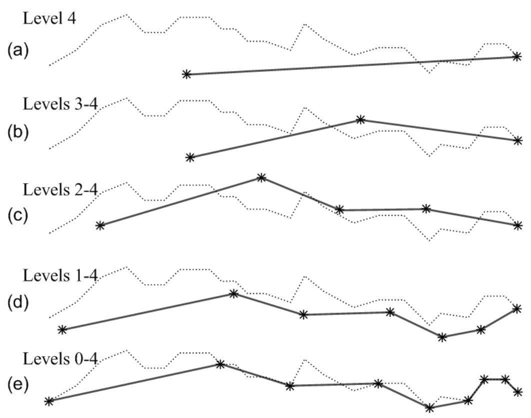

Fig. 5. Multiple resolutions of a trajectory.

tion process will produce $H + c + 1$ points, because there are $H + c$ consecutive segments in that approximation. This simple calculation gives 6 points for the state at $\tau = 15$ (Fig. 3a), while it yields 9 points when $\tau = 22$ (Fig. 3h).

4.4 Multi-resolution Trajectory Approximation

The aforementioned bottom-up approach will retrieve all available positional data stored in an AmTree. Instead of such an exhaustive process, we will describe another reconstruction policy that can produce amnesic trajectory approximations at multiple resolutions. Clearly, trajectory segments stored in successive tree levels are connected to each other. Therefore, if we take any number of the higher levels of the tree, we can still reconstruct an uninterrupted representation of the movement for the corresponding time intervals. Essentially, we still apply the bottom-up technique of Section 4.3, but for a restricted number of levels.

If this process is repeated H times, each time getting another level lower in the hierarchy, we will be able to get multiple trajectory approximations, at progressively finer resolutions for the most recent portions, as illustrated in Fig. 5. The initial approximation is performed using only the upmost level of the tree depicted in Fig. 3h. Since the point at hand is the last seen location, we get the other point from the displacement available in node R_4. Note that, since the tree is not entirely synchronized at this timestamp, this procedure will not yield the actual location of the moving object 16 timestamps ago (Fig. 5a). The second approximation will include information from level 3 as well, and will be slightly better (Fig. 5b). The final approximation (Fig. 5e) retrieves all information currently summarized in the tree and the locations it yields are the original positions at the respective timestamps.

Multi-resolution reconstruction is application dependent, enabling fast estimations from coarse trajectory representations. Critical features (e.g., movement orientation) can be quickly conjectured by inspecting just a few upmost levels.

5 Computing Aggregates over Moving Objects

In this section, we present a summarization technique that provides unbiased estimates about the number of objects moving in an area of interest during a specified time interval. When each object must be counted only once, the problem is known as *distinct counting* [12]. We introduce an amnesic treatment for this problem, utilizing the AmTree structure with a powerful sketching algorithm over a static spatial decomposition.

5.1 Overview of Flajolet-Martin (FM) Sketches

FM sketches [6] were designed to approximate the zeroth frequency moment (F_0), which actually provides an estimate for the number of *distinct objects* in a multiset. They have since widely used in stream summarization (e.g., [7, 12]).

As described in [12], an FM sketch is a bit-vector consisting of r bits, all of them initially set to 0. The appropriate size of this bitmap is $O(\log_2 DC)$, where DC is an upper bound on the number of distinct objects. FM algorithm employs a hash function h that maps an object identity oid into a pseudo-random integer $h(oid)$ with a geometric distribution, meaning that $Pr[h(oid) = v] = 2^{-v}$, where $v \in [1, r]$. For every object oid in the multiset, the algorithm sets the $h(oid)$-th bit of the FM sketch to 1. Regardless of how many times object oid is found in the set, the same bit will always be switched on; this fact establishes the duplicate-insensitivity property of FM sketches. No matter in which order objects will appear, the same bitmap will be eventually created. For n distinct objects, it is expected that $n/2$ of them will map to bit 1, $n/4$ objects go to bit 2, and so on.

It turns out that, if we locate in the sketch vector its first bit k that is still 0, we can get a good unbiased estimate of n as $1.29 \cdot 2^k$. To improve probabilistic confidence, Flajolet and Martin [6] suggested the use of m independent sketches, each with its own hash function, and finally averaging all results to estimate n. Furthermore, to keep processing cost per object to $O(1)$ instead of $O(m)$, Probabilistic Counting with Stochastic Averaging (PCSA) was proposed, which utilizes m atomic FM sketches. Every time an update is due, only one of these sketches is chosen using another hash function $h'(oid)$. Thus, every sketch keeps count for about $\frac{n}{m}$ distinct objects. Assuming that $k_1, k_2, \ldots, k_m$ are the first non-zero bits in each of the m atomic sketches, the distinct count is estimated to $n = 1.29 \cdot m \cdot 2^{\frac{1}{m} \sum_{i=1}^{m} k_i}$, with expected standard error $O(m^{-\frac{1}{2}})$ and overall space complexity $O(m \log_2 DC)$.

Another important property of FM sketches is that they can be *composed* from other partial FM sketches. If a sketch FM_A is maintained over multiset A and a sketch FM_B is independently computed over multiset B, we can get the FM sketch of $A \bigcup B$, by applying a bit-wise OR over their bitmaps (and between respective sketches for the PCSA variation), i.e., $FM_{(A \bigcup B)} = FM_A \vee FM_B$.

5.2 Applying 3-tier Compression

We consider queries of the form *"How many distinct moving objects have been observed in area α during time interval $\Delta\tau$?"*. In our terminology, α is the *spatial*

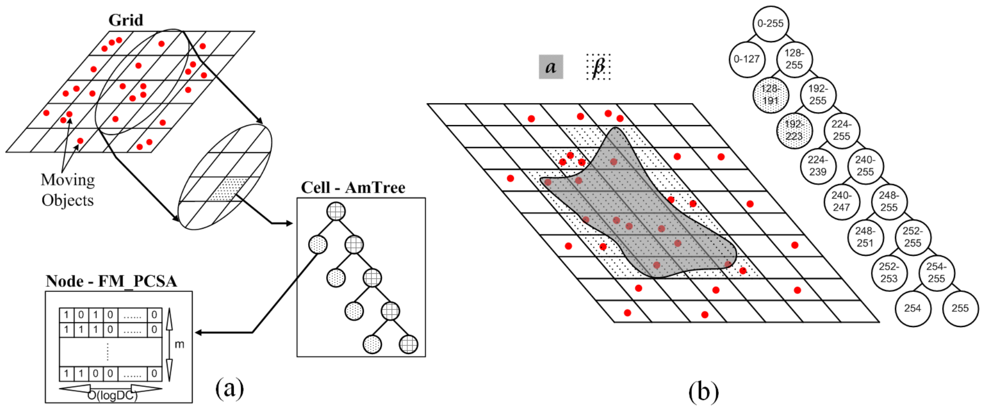

Fig. 6. (a) The 3-tier FM-AmTree structure. (b) Evaluating a range query.

extent of the query region defined as a 2-d polygonal area, while $\Delta\tau$ is the *temporal extent*. The distinguishing feature of our approach is that we allow an amnesic behavior to the resulting estimations: as long as the temporal extent remains close to the current time, it is probable that certain time intervals stored in the AmTree can approximate it with more precision. To this end, we introduce a 3-tier structure that accomplishes compression at three levels (Fig. 6a):

1. We utilize a regular decomposition of the 2-d spatial plane into equal-area cells. If HG and VG are respectively the number of subdivisions along axes x, y of this grid partitioning, then a set of $HG \cdot VG$ cells is used to maintain a simplified spatial reference of moving objects instead of their actual locations.
2. To accommodate temporal extents, each cell points to an AmTree, which maintains gradually aging count of objects within that cell.
3. Query-oriented compression is achieved using FM_PCSA sketches. Each node of an AmTree corresponds to m bitmap vectors utilized by the FM_PCSA sketch. Hence, we avoid enumeration of objects, as we are satisfied with an acceptable estimate of their distinct count given by the sketching algorithm.

Assuming that N locations have been received for each object and that DC is an upper bound of the distinct objects, it is easy to see that the overall space complexity is $O(HG \cdot VG \cdot \log N \cdot m \cdot \log DC)$. Beyond its small memory footprint, this technique is well suited to a streaming context, due to its low update cost.

Indeed, when a positional update of an object arrives for processing, initially its location (x, y) is hashed against the spatial grid and the appropriate cell is identified. Then, the FM sketch linked to node R_0 of the corresponding AmTree must be updated. This is carried out by hashing the object's id over this FM sketch. When updates must propagate to higher levels of the tree, merging function g takes advantage of the ability to compose partial FM sketches into a single sketch. Thus, the union of sketches $FM_{R_i} \bigcup FM_{L_i}$ at the same level i is assigned to node R_{i+1}. For synchronization, AmTree updates are performed when all point locations for the current timestamp τ have arrived, assuming that each object discloses its position at every τ. In total, $HG \cdot VG$ trees need to be updated at each timestamp, so update complexity is $O(HG \cdot VG)$ per timestamp.

5.3 Estimating Count of Distinct Objects

In order to estimate the number of distinct objects moving within area α during time interval $\Delta\tau$, we initially start by identifying which cells of the grid partitioning completely cover region α. As shown in Fig. 6b, the cells overlapping with polygon α constitute a greater area β. Next, those cells are used to determine the group of qualifying AmTree structures that maintain the aggregates. Corresponding nodes at every AmTree actually refer to identical time intervals, since all trees are concurrently updated. Thus, we need to locate the set of nodes that overlap time period $\Delta\tau$ specified by the query; these nodes are the same for each qualifying tree. As illustrated in Fig. 6b, when the AmTree snapshot is traversed, nodes L_5 of interval [192..223] and L_6 of [128..191] are found to make up the minimal time period [128..223] that contains the query interval $\Delta\tau = [135..220]$. This means that we should access only the FM sketches linked to these two nodes in each of the AmTrees qualifying for area β. By taking the union of all those sketches (i.e., an OR operation over the respective bitmaps), we finally get an approximate answer to our query.

Clearly, the response given by this algorithm is an overestimation of the actual distinct count, as it is influenced by the approximations performed in each tier. First, the degree of grid partitioning into cells apparently controls the spatial resolution of aggregates. Second, the temporal granularity determines the aging rate inherent in the AmTree mechanism, and consequently the extent of time intervals maintained. Last but not least, the number of bitmaps in each sketch affect the precision of the estimated counts. It goes without saying that there is a trade-off between processing time and each of these three parameters that guide response accuracy, a fact empirically confirmed in the experiments.

6 Experimental Evaluation

Next, we report on an experimental validation of AmTree when utilized (i) to approximate trajectories of moving objects (Section 4) and (ii) to provide estimates for distinct count queries (Section 5), always working in main memory.

All experiments were performed on an Intel Pentium-4 2.5GHz CPU running WindowsXP with 512 MB of main memory. We generated synthetic datasets for trajectories of objects moving at various speeds along the road network of greater Athens (an area of about 250 km^2). Geometric nodes of the road segments were randomly chosen as origin and destination points for a shortest path algorithm. Finally, 500 sample points were taken for each calculated route.

We run simulations for several time intervals (up to 500 timestamps) and number of objects (up to 20,000). The interarrival time of streaming tuples was constant for all trajectories, assuming that all objects reported their positional updates concurrently at regular time intervals. This is actually the most intensive situation, since agility of objects was set to 100%.

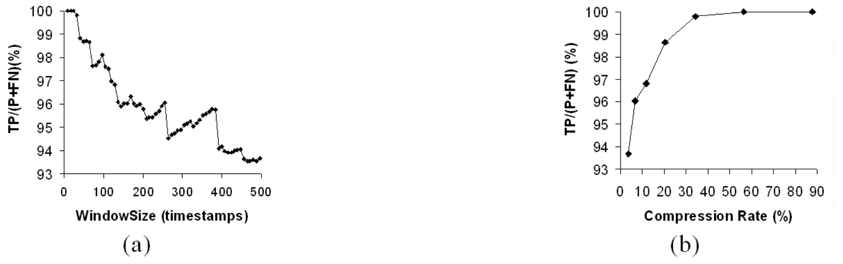

Fig. 7. Approximation quality for trajectories.

6.1 Quality of Trajectory Approximation

In this set of experiments, we maintained an AmTree structure for each trajectory from a set of 1000 moving objects, each recorded for 500 timestamps. To assess approximation quality, we utilized spatiotemporal range queries that continuously return objects contained in a given spatiotemporal hyper-rectangle. We defined 10 static spatial rectangles, each covering about 1% of the total space, and at every 8 timestamps we observed which approximate trajectories fell within the spatial area. Hence, temporal extents proceed by 8 timestamps each time, so that each window spans the entire movement history thus far.

We counted qualifying trajectories in each query region and results were averaged for each temporal interval. Then, we compared these estimations against exact answers based on original trajectories. We measured approximation quality as $\rho = \frac{TP}{P+FN}$, where TP denotes the number of *true positives*, P is the number of (false and true) *positives*, whereas FN signifies the number of *false negatives*.

We calculated values of ρ as plotted in Fig. 7a, where the x-axis of the diagram shows the query intervals used, i.e., the size of landmark windows. As expected, the quality of approximation deteriorates as the query interval becomes gradually larger due to the time-decaying positional accuracy of the trajectory segments. As stated in Section 4.3, the most remote trajectory segments progressively obtain an increasingly rough outline maintaining a reduced number of the original points. Although recent segments are always more accurate, overall error heavily depends on the temporal extent in the past, since we steadily get away from the "landmark" time instant the query was issued.

Notice that abrupt drops in approximation quality occur at timestamp values that correspond to major updates in AmTree, i.e., changes across the majority of tree levels. When all nodes get synchronized, segments from the right nodes suffice to provide all information contained in the tree. Subsequently, changes occur at waves that always propagate to a higher level until a complete update happens again. Recall from Section 4.3 that, at snapshots where more tree levels have been updated at diverse waves, the number $H+c$ of reconstructed segments has a local maximum, so we can get more additional points by taking advantage of information stored in left nodes. This is reflected in the sporadic "spikes" in Fig. 7a that signify trajectory representations with maximal number of points.

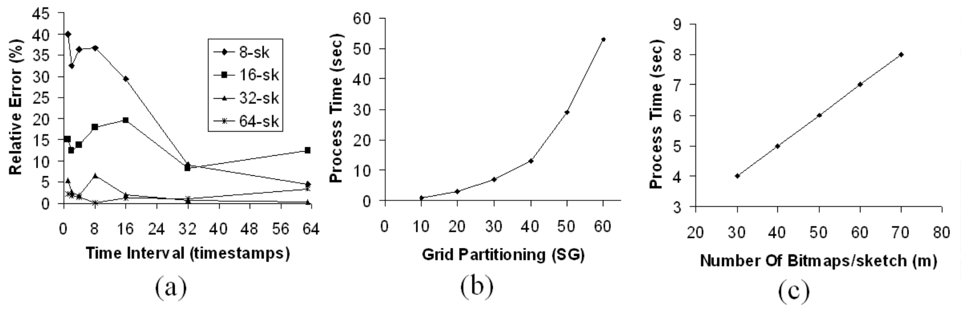

Fig. 8. Quality of aggregate estimation.

Fig. 7b shows approximation quality versus the compression rate achieved. Compression rate denotes the ratio in the size of an approximated trajectory with respect to the original, i.e., the percentage of original points retained in the coarsened trajectory. Not surprisingly, approximation quality decreases for more compressed trajectories, since more points are discarded. As the temporal extent gradually increases, the amnesic behavior of AmTree gets more pronounced and a trajectory obtains exponentially more compressed representation.

Still, even for heavily compressed trajectories (rates close to 2%), accuracy index ρ is quite satisfactory (about 93.5%). For small compression rates, the additional deviation is attributed to trajectory segments far away from the current time, i.e., those that have been consistently reduced over and over again, eventually devoid of much positional information. In contrast, answers reconstructed from most recent time intervals are remarkably more accurate, since the respective segments have not been altered much yet.

6.2 Quality of Distinct Count Estimates

To validate the algorithm for estimating aggregates, we utilized a dataset containing locations of 20,000 objects, each recorded for 100 timestamps. We assumed square-shaped grid cells, so the degree of spatial decomposition SG is a single parameter for both axes x, y, i.e., $SG = HG = VG$.

In Fig. 8a we illustrate the effect of the number m of bitmaps per FM_PCSA sketch on relative error, with respect to distinct count queries for various temporal extents, fixing $SG = 10$. Quite predictably, the more bitmaps allocated to each sketch, the more confidence is attained for the estimates. While for $m = 8$ estimation quality was off even by 40%, for $m = 64$ the relative error was less than 4%. Further increase in m did not seem to yield better results.

In Fig. 8b we plot the total update cost against the spatial decomposition SG at each dimension, for $m = 32$ skecthes. Note that the cost is quadratic to the number of cells, i.e., a finer grid partitioning incurs more processing time at the expense of increased accuracy. Besides, computation over the entire stream of locations (and the update cost per tuple as well) is linear to the number m of sketch bitmaps, as depicted in Fig. 8c, when $SG = 10$.

7 Related Work

The notion of time-decaying significance in data items is widespread in streaming applications, such as telecom usage patterns or connections through Internet gateways. Aging-aware computation of aggregates and statistics over a stream was first formalized in [5]. Several types of time-decaying functions were defined and theoretically analyzed in terms of their storage requirements to fine-tune the rate of decay, considering exponential, polynomial and sliding-window variants.

Amnesic approximation of streaming time series was set forth in [9], offering the intuitive idea that data can be approximated with a precision proportional to its age. A taxonomy of amnesic functions was provided, along with online algorithms that can incrementally compute reduced representations of scalar time series with time complexity independent of the total stream size. Our approach falls naturally under this generic framework, as an amnesic structure with exponential decay especially tailored for streaming positional updates.

AmTree scheme extends the time-varying SWAT mechanism proposed in [3], which produces multi-resolution approximations of a data stream. Despite its external similarity to SWAT, AmTree is more generic in certain aspects. First, SWAT is intrinsically bound to wavelet transform of scalar stream values, whereas AmTree can even handle multi-dimensional points with user-specified approximation functions. Further, in functionality terms, at each tree level we eliminate the intermediate node, which SWAT utilizes for maintenance of transient values. Finally, we can effectively tune approximation accomplished at each level, by properly controlling the number of granules (i.e. tree nodes).

A multi-resolution tree structure was introduced in [8] for computing aggregates (SUM, COUNT, etc.) over multi-dimensional points, by employing typical spatial indices (e.g., quad-trees, R-trees). Still, this technique cannot cope with streaming data that arrive at high rates, while it cannot easily accommodate the sequential nature of positional updates. Temporal aggregation over streams was also studied in [13], but the emphasis was mainly on indexing schemes that could be dynamically maintained using multiple time granularities. Efficient exact computation of aggregates over sliding windows is tackled in [1] with a particular interest in sharing resources for numerous continuous queries over scalar stream values. In contrast, our concern is to maintain age-biased synopses to get reduced representations, rather than just computing aggregate statistics.

The distinct-count problem over moving objects was tackled in [12]. Their solution applies to spatiotemporal databases and builds a sketch index [6] equipped with R-trees to maintain spatial regions and B-trees to host temporal intervals. This scheme maintains all successive stages of aggregation without any notion of time-decay; instead, our 3-tier approach is geared towards a streaming model.

In [10] we proposed two single-pass sampling techniques to achieve effective trajectory compression, exploiting spatial locality and temporal timeliness inherent in trajectory streams. The underlying principle of those heuristics was that speed and orientation of each object play an important role in deciding which samples to retain. Departing from this data-driven approach, in this paper we elaborate on our ideas briefly outlined in [11]. Thus, we focus on aging-aware schemes where all stream tuples have a lifespan, instead of discarding data upon admission to the system as carried out with a sampling technique.

8 Conclusions

In this paper, we presented a hierarchical structure called AmTree, capable of maintaining multiple-granularity approximations over streaming locations of numerous moving objects. This method works in an online amnesic fashion, by reserving more precision for the most recent items while accepting increasing error for gradually aging stream portions. We empirically confirmed that this time-decaying approach can effectively cope with online trajectory approximation, also providing affordable estimates for distinct count spatiotemporal queries.

In the future, we intend to study other types of amnesic behavior adjusting this structure to deal with user-specified aging patterns. In terms of answering queries over positional streams, we plan to explore spatiotemporal associations, such as the aging rate of query intervals or topological relations between moving objects and query regions, in order to better assess approximation quality.

References

1. A. Arasu and J. Widom. Resource Sharing in Continuous Sliding-Window Aggregates. In *VLDB*, pp. 336-347, September 2004.
2. C. Bettini, C.E. Dyreson, W.S. Evans, R.T. Snodgrass, and X.S. Wang. A Glossary of Time Granularity Concepts. *Temporal Databases: Research and Practice*, LNCS 1399, pp. 406-413, 1998.
3. A. Bulut and A.K. Singh. SWAT: Hierarchical Stream Summarization in Large Networks. In *ICDE*, pp. 303-314, March 2003.
4. Y. Chen, G. Dong, J. Han, B. W. Wah, and J. Wang. Multi-Dimensional Regression Analysis of Time-Series Data Streams. In *VLDB*, 323-334, August 2002.
5. E. Cohen and M. Strauss. Maintaining Time-Decaying Stream Aggregates. In *PODS*, pp. 223-233, June 2003.
6. P. Flajolet and G.N. Martin. Probabilistic Counting Algorithms for Database Applications. *Journal of Computer and Systems Sciences*, 31(2):182-209, 1985.
7. S. Ganguly, M. Garofalakis, and R. Rastogi. Processing Set Expressions over Continuous Update Streams. In *SIGMOD*, pp. 265-276, June 2003.
8. I. Lazaridis and S. Mehrotra. Progressive Approximate Aggregate Queries with a Multi-Resolution Tree Structure. In *SIGMOD*, pp. 401-412, May 2001.
9. T. Palpanas, M. Vlachos, E. Keogh, D. Gunopulos, and W. Truppel. Online Amnesic Approximation of Streaming Time Series. In *ICDE*, pp. 338-349, March 2004.
10. M. Potamias, K. Patroumpas, and T. Sellis. Sampling Trajectory Streams with Spatiotemporal Criteria. In *SSDBM*, pp. 275-284, July 2006.
11. M. Potamias, K. Patroumpas, and T. Sellis. Amnesic Online Synopses for Moving Objects. In *CIKM*, pp. 784-785, November 2006.
12. Y. Tao, G. Kollios, J. Considine, F. Li, and D. Papadias. Spatio-Temporal Aggregation Using Sketches. In *ICDE*, pp. 214-226, March 2004.
13. D. Zhang, D. Gunopulos, V.J. Tsotras, and B. Seeger. Temporal Aggregation over Data Streams using Multiple Granularities. In *EDBT*, pp. 646-663, March 2002.