Evaluating Stream Predicates over Dynamic Fields

J.C. Whittier Qinghan Liang Silvia Nittel
School of Computing and Information Science
University of Maine, USA
{john.c.whittier}@maine.edu, {qliang, nittel}@spatial.maine.edu

ABSTRACT
Technological advances have created an unprecedented availability of inexpensive sensors able to stream environmental data in real-time. However, we still seek appropriate data management technology capable of handling this onslaught of sampling in previously unavailable spatial and temporal density. Data stream engines (DSEs) are state of the art data management tools that have update throughput rates of up to 500k tuples/s. In previous work we have shown that DSEs can be extended to generate smooth representations of continuous spatio-temporal fields sampled by up to 250K sensors on-the-fly in near real-time, creating a new representation every second. In this paper we investigate a spatio-temporal stream operator framework that can efficiently execute predicate operators over such spatio-temporal fields. Typical predicates are e.g. "find all sub-areas in a field that are below or above a certain threshold value". We present the requirements, the approach taken, and our results along with a performance evaluation.

Categories and Subject Descriptors
H.3.3 [**Information System Applications**]: Spatio-temporal systems – *data streaming.*

General Terms
Data stream engines, spatio-temporal predicate, algorithms, performance.

Keywords
Data streams system, sensor data streams, fields, continuous phenomena.

1. INTRODUCTION
Real-time data streaming with sensors is used in applications such as intrusion monitoring, manufacturing, traffic management, disaster response, radioactive accidents, air quality control, pollen monitoring and other types of sensor networks [5, 24, 25]. Thousands of sensors are distributed over a geographic area and connected directly to the Internet to collect high-frequency updates [19]. With this technology we can also monitor *continuous* phenomena over e.g. a metropolitan area in real-time, continuously over space and time. Individual sensors can only deliver point-based, raw sensor data streams, but humans more intuitively perceive integrated, continuous representations of continuous phenomena. However, current geographic information systems are unable to handle the permanent high update load of thousands of sensors and on-the-fly integrate these streams into representations such as a TIN or raster every few seconds.

5th Int' Workshop on Geostreaming, Nov 4, 2014, Dallas, TX, USA.

In previous work [22, 28], we have shown that commercial data stream engines (DSEs) such as Microsoft Streaminsight, IBM's Infosphere, Oracle's CEP or Streambase [2, 4, 27, 29] are viable systems for dealing with continuous high throughput of very large numbers of concurrently streaming sensors. We developed a DSE architecture extension consisting of a novel spatio-temporal stream operator framework for dynamic continuous phenomena (ST-CPF) [22, 28]. This scalable framework consists of a set of stream-based, main memory query operators that are connected via queues, and generate raster representations of a continuous phenomenon on-the-fly. ST-CPF contains a main memory, spatio-temporal grid index and an efficient kNN-based algorithm of spatio-temporal inverse distance weighting (ST-IDW). ST-CPF achieves a throughput of updates of up to 250K sensors with individual sensor update rates of 1 update/s, generating new raster representations every few seconds. With such a framework, it is possible to observe a dynamic phenomenon continuously over both space and time in near real-time.

For continuous analysis of such phenomena, also called fields, threshold operators are key. A threshold operator retrieves spatio-temporal subregions of a field in which the field's values meet a user-set threshold (e.g. wild fires in a temperature field, or concentration values in a toxic plume). More mathematically, such an operator is a predicate. A predicate is expressed as a condition over the values of a continuous phenomenon, and elements that evaluate to true are elements of the result set. The condition can target the spatial, temporal or thematic component of a phenomenon. For example, the following query contains a thematic predicate retrieving the radiation values satisfying a condition within the boundary of Japan (the result is rendered as a raster):

*SELECT RASTER(**Predicate**(ST-IDW(@Grid, s.loc, s.val, @IC), @**Condition**)) as high_radiation_deposition*
FROM sensors s WINDOW 5min
WHERE s.sensor_type=radiation AND INSIDE(s.loc, @Japan)

Predicates over continuous phenomena are common stream queries. Evaluating such predicates over continuous phenomena relies on point-based sensor streams. In reality, a continuous environmental phenomenon such as the dispersion of radioactive particles over a region is seamlessly distributed over a geographic region, and gradually changes over space and time. There are typically no 'hard' boundaries. So, how do we evaluate the predicate result *most accurately*? Here, a phenomenon's continuous representation has to be generated on-the-fly and the predicate computed. However, the predicate result is potentially a small subregion of the entire phenomenon, and generating entire representations is computationally expensive. The question is how to evaluate predicates in sensor stream environment *efficiently*? Thirdly, the predicate result region(s) changes continuously over space and time as the region(s) expands, shrinks or simply moves. How and in what situations can we use these characteristics to optimize predicate evaluation using incremental evaluation?

Our contributions: In this paper, we present several approaches to evaluating predicates over spatio-temporally continuous phenomenon efficiently. In a naïve method, which we use as the

baseline, a field is represented as a raster, and all cells of the entire raster are interpolated within each query window. Subsequently, each cell is filtered based on the predicate condition. We developed two novel approaches that start with the predicate condition to determine which subregions of the field need be interpolated, i.e. the *region growing* and the *tile-based approaches*. The key to these methods is that a pre-filter generates *seed cells* that are likely included in the predicate results. In the region growing approaches the seeds are grown into regions. With a high-resolution field representation, this can result in significant runtime and memory cost due to the very large number of grid cells. To counter this cost, we also develop the tile-based approach. Here, raw sensor samples are filtered to identify '*seed tiles'* (a group of cells) instead of seed cells, which are then expanded. Furthermore, we investigated an *informed, incremental predicate evaluation strategy*. Here, the evaluation of a new query window starts with the result of the previous window as seeds, and makes incremental updates to determine which cells will be elements of the predicate result, and which not. The performance results show that the informed method outperforms the un-informed tile expansion method if a phenomenon is slowly changing over time and the area is large.

The remainder of this paper is structured as follows: Section 2 gives a short overview of our previous work, which is the background for this article. In Section 3, we state the problem. In Section 4 and 5, we present and discuss evaluation strategies for predicates over dynamic phenomena. Section 6 contains the performance evaluation. Section 7 discusses the related work, and Section 8 summarizes the results and discusses future work.

2. Previous work

2.1 Data stream engines

A data stream engine (DSE) is a software system designed to manage unbounded data streams such as those found in real-time financial, emergency or intrusion monitoring applications. Data streams are continuous and the amount of data arriving at a particular time is unbounded. Further, updates per entity (e.g. a sensor or a stock) are often extremely frequent. A generic sensor data packet can represented as a tuple or a record which contains attributes such as *<sensor id, sensor type, timestamp, location, value measured>*. In traditional database systems (DBSs), queries are evaluated by *pulling* data from disk, while in DSEs queries are evaluated over data *pushed* from stream sources. Additionally, traditional DBS queries are executed one time, whereas DSE queries are continuously re-evaluated and potentially run indefinitely. This means that new strategies had to be developed to handle these challenges:

Continuous Query Model: Traditional DBS queries operate on a finite data set (i.e. a relation) and assume set-based data, i.e. they do not have to consider the order of data. In contrast, continuous queries operate indefinitely on an unbounded data set and take the *temporal order* of arriving data into consideration using an additional specification for query evaluation intervals, also called query windows [3].

Low Latency: Data from streams is generally critical for answering real-time queries, but its significance is often short-lived.

Variable data rate: The data arrival rate for a stream-based application could vary from hundreds to millions of updates per second. Additionally, the arrival rate can be unpredictable given fluctuations in the sensor update rates or transmission delays. Irrespective of the bottlenecks, DSE query processing must be robust and aimed towards high throughput.

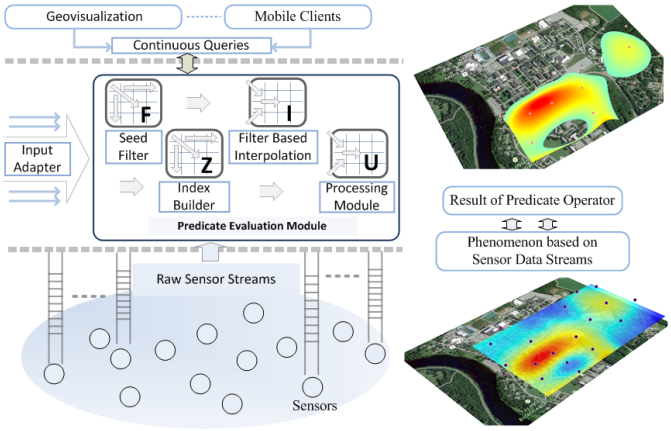

Figure 1 Overview

While the structure of tuples in a DSE is similar to that of relational tuples in a DBS, query evaluation in a DSE is vastly different. Since a *stream* is an unbounded sequence of time-stamped tuples, query windows are used to generate a finite scope for query execution; query windows which partition the stream into finite sets of tuples. Under a non-blocking stream processing paradigm, a query over streams consists of stream operators arranged as a *directed acyclic graph* (DAG). The operators are connected via queues, and each stream operator has an in-memory state consisting of any tuples necessary to perform its operation. For high throughput, all data is stored exclusively stored in main memory, including index structures. Tuples and indices are often shared between operators. Stream operators execute in parallel; by using techniques like operator cloning, query performance can be scaled up without change to the operator graph itself [21].

Today several commercial DSEs such as Oracle CQL [23, 29], Microsoft Streaminsight [1, 2], IBM Infosphere [4], and Streambase [27] are available. In most commercial systems, the stream option is an extension of the relational DBMS product, and libraries that offer spatial support are often available for programming stream queries. However, related work shows that most existing spatial library functionality was *not* implemented as non-blocking stream operators and instead adheres to the traditional disk-based processing paradigm, creating a significant performance bottleneck [1, 11, 14].

2.2 Extending DSE for continuous phenomena

To monitor continuous phenomena, a very large number of individual sensors stream their updates to a DSE. To understand the phenomenon, e.g. a field of pollen or radiation deposition, users find it more intuitive to interact with a *continuous* representation of the phenomenon instead of individual sensor readings.

We have developed a DSE architecture extension called the spatio-temporal continuous phenomena framework (the ST-CPF). ST-CPF consists of a scalable, extensible stream operator framework that generates a continuous and smooth phenomenon representation based on available sensor streams. We make the realistic assumption that individual sensors send samples with varying update frequency (intra and inter stream variability) and are not update synchronized. Additionally, device locations can change. The ST-CPF framework processes dynamic phenomena such as pollen fields, radiation or air pollution in temporal 'portions' defined by the query window over the input sensor data streams. Incoming sensor samples resemble a spatio-temporal point cloud, and are cached in a pane-based spatio-temporal grid index [28]. ST-CPF also includes stream based implementations of a spatio-temporal Inverse Distance Weighting method (ST-

IDW) [15, 28], which takes a spatio-temporal point cloud as input to approximate and fill in non-sampled points in space and time. Other ST interpolation methods are possible, and the specific choice depends on the application needs. To approximate the 'reality' of a phenomenon *correctly* over time and space within the query window, we introduced the concept of an *interpolation center (IC)*; the IC is a chosen time stamp within the query window and is used to create an approximation of the state of the dynamic phenomenon at that particular point in time. Samples surrounding the interpolation center both in space *and* time serve as input for ST-IDW to generate a representation of the predicted state of the phenomena at that time instant. Updates that are in greater temporal distance from the interpolation center have less impact than samples collected temporally closer to the interpolation center (similar to spatially close samples having more weight on the prediction). Thus, we weigh samples based on their spatial and temporal nearness. The query result is a 2D grid representation of the phenomenon over the specified observation region. In real-time monitoring applications, users are mostly interested in the *latest state* of a phenomenon, so the IC is defined at the query window end. This has a *time decaying* effect on the weight of older samples (beginning of window).

While it is useful to generate a representation of the entire field for visualization purposes, predicate operator over fields are basic operators for continuous analysis of fields.

3. PREDICATES OVER FIELDS

In this section we define predicates over fields and discuss the challenges of evaluating predicates over dynamic fields.

3.1 Dynamic fields

To represent the real-time and dynamic prediction of a continuous spatio-temporal phenomenon, the concept of a *field* provides a mathematically defined abstraction [9, 12, 13]. A field is a function from each position in the field's domain to an attribute value in the field's range, i.e. $f: \vec{P} \rightarrow A$. If the position p consists of a spatial point (x, y) and a timestamp t, we call the field a *dynamic field* since it represents the field's continuous changes over space and time. In practical terms, a field f allows us to retrieve or calculate a value for each location and time in the field's domain. While a mathematic field definition yields an appropriate concept to model a geographic entity, once such a field is presented on a computer it requires a finite and discrete representation. The field consists of a number of preset spatio-temporal positions, and the requested values for each position are calculated based on the available sensor streams combined with results from a spatio-temporal interpolation method for all non-sampled points. Fields allow more flexibility for the calculation of predicates over dynamic fields since the resolution can be adjusted to the predicate result.

3.2 Predicates for dynamic fields

A field predicate is a unary field operator; a Boolean proposition is applied to each element of a field's domain.

$$r = P_{prop}(f)$$

We define that the field operator is closed over the set of fields; thus, it takes a field f as input and yields a field r as a result. For example the predicate $P_{temperature>0}$ applied to a temperature field f results in the field r. How can we define the result field r? Intuitively, r should be a temperature field since it is derived from a temperature field. Thus, all positions within r's domain are mapped to values. However, the domain of r, $dom(r)$, is a subset of $dom(f)$ because it is limited to positions at which the Boolean proposition is true. Additionally, $dom(r)$ can potentially change with each query window evaluation.

While predicates over space, time and/or values are relevant, in this paper we solely focus on the evaluation of predicates over values. Such a predicate can be thought of as "find all values that meet a certain condition and return a subfield for each query window". For example, take the following query:

*SELECT RASTER(**Predicate**(ST-IDW(@Grid, s.loc, s.val, @IC), @**Condition**)) as high_radiation_deposition*
FROM sensors s WINDOW 5min
WHERE s.sensor_type=radiation AND INSIDE(s.loc, @Japan)

This example stream query requests subfield areas over Japan at which a predicate condition is fulfilled; the condition can be that the caesium-137 activity concentration at each point is greater than 1 Bq/m^3 (Becquerel/m3). The predicate is evaluated over a raster representation of the field, which is interpolated using ST-IDW (with given grid size and interpolation center). Each query window returns a new raster snapshot of the predicate result, and the result field (likely) gradually changes with each snapshot (see Figure 2).

We observe that the actual chosen computer representation of the field does have an impact on the accuracy of the predicate. In the mathematical (ideal) presentation, changes between neighboring values are gradual. Once represented as a grid, cell values can be checked against the predicate condition and return true or false. Using field representations such as rasters or TINs, the representation and its resolution has an impact on the predicate result *precision*. Further, the predicate operator needs to calculate the result based on available input point samples and use spatio-temporal interpolation for the rest of the subfield to actually capture correct regions.

3.3 Incremental evaluation of predicates

In general, under ST-CPF a predicate is evaluated as follows: *first*, all incoming sensor tuples for a query window are organized by space and time into a shared spatio-temporal pane-grid index (for detailed information, see [28]); *secondly*, the predicate is evaluated interpolating each cell using ST-IDW with a user-defined interpolation center. Then, the interpolated value is tested to determine if it satisfies the predicate or not. If it does, the cell is added to the predicate result set. We use an efficient, adaptive k Nearest Neighbor (akNN) based ST-IDW implementation [28]. Here, for each interpolated cell, tuples from successively more distant cells (distance in both time and space) are used to predict the cell value, but when a specified number of tuples have been received, the interpolation for the cell is completed. This method provides a uniform interpolation result capable of coping with irregular sampling distribution and density. A new predicate result is returned for each query window, and evaluation is incremental, scalable, and adaptive as all operators and data are main memory based.

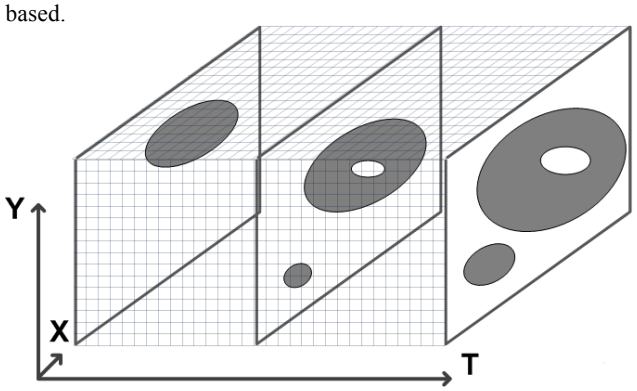

Figure 2 Predicates over dynamic fields

In a naïve predicate evaluation algorithm, *all* cells of the raster are interpolated and filtered after the interpolation step. However, an efficient evaluation strategy depends on the percentage of the field that is part of the predicate result: if a large part of the entire field evaluates to true, the naïve algorithm should work well. However, if only a relatively small percentage of the field qualifies it is more efficient to identify these cells, reduce caching of samples, and limit the spatio-temporal interpolation to the result subfields. We focus on this case for the remainder of this article but compare these algorithms with the naïve algorithm in the performance section.

4. SEED-BASED PREDICATE EVALUATION

While our previous work focused on the continuous query processing of high-throughput based ST interpolation of entire fields, we focus on the aspect of efficient predicate evaluation over *small* subfields in this paper. Once the set of cells that are likely part of the predicate result set is identified, ST-interpolation of each cell is similar to our previous work [28].

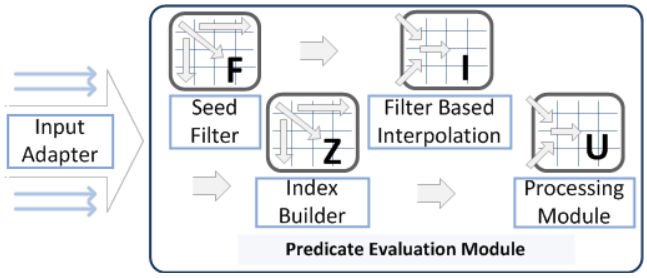

Figure 3 Stream operators for predicate evaluation

4.1 Basis considerations

A value predicate applied to a geographic phenomenon typically retrieves features. For example, the predicate $P_{temperature>1000}$ applied to a temperature field would identify fires within an observation region. Characteristically for such features, the sets of cells that evaluate to true are spatially and temporally contiguous. This fact can be exploited within an evaluation algorithm by identifying 'seeds' of such regions. By expanding seeds spatially and temporally, the algorithm can potentially reduce the search for grid cells that need to be checked against the predicate.

First, 'seeds' of such result regions need to be identified. In ST-CPF the predicate evaluation stream operator graph splits the incoming sensor streams updates into two operators: the existing *grid-pane index builder* and the new *seed filter* (see Figure 3). The seed filter checks all incoming tuples based on the predicate condition. If a *tuple* meets the condition the *cell* in which it is located is added to a 'set of seeds'. Although single tuples may meet the predicate condition, it is not clear whether the entire cell once interpolated will meet the predicate condition. Their participation in the predicate result can only be determined after using all available samples in that cell and potentially some surrounding cells during ST-interpolation.

In this predicate evaluation algorithm, *organizing seed cells* is a crucial step towards efficient predicate evaluation. As mentioned, seed cells tend to be spatially clustered; however, the incoming order of potentially millions of sensor updates is *random* and *ordered by time*. Filtering tuples based on the predicate condition also creates seeds cells in random order. One option is to use a data structure that organizes seed cells within a query window spatially so that clusters of cells can be derived more easily. A candidate data structure is a main memory-based spatial index such as a quadtree, since its behavior does not deteriorate under such massive update load and tree traversal can identify contiguous regions of grid cells that are candidates for interpolation. However, the segmentation of quadtrees into quadrats and a potential division of candidate regions into disconnected quadrants makes identification of relevant regions more computationally expensive.

In our proposed approaches, the data structure for seed cells is hashmap keyed on the (x,y) location of each seed, the insertion order of cells into the set of seeds follows the random order of tuples in the stream. The different methods for 'expansion' of seed cells into regions representing the predicate result are designed to minimize the search and test effort.

4.2 Alternative 1: Region growing

The first alternative algorithm is a region-growing algorithm. Seed-based region-growing algorithms are common in image processing and feature extraction algorithms. Similar to predicates, region-growing algorithms traverse a grid and test cells based on their value. The difference is that in the case of dynamic phenomena, grid cells need to be interpolated *before* the value can be tested. Further, all data stored is main memory and performance demands are near real-time. We will discuss two different region-growing algorithms: Breadth First (BF) and Scanline (SL).

4.2.1 Breadth First

Seed Selection: A tuple's (x,y) coordinate values are converted to integer (x,y) cell coordinate values (center of the cell), and a new pair <x,y> containing the new seed coordinates is added to the queue of seeds. The seed represents a grid cell containing at least one value satisfying the threshold predicate. If the coordinate already exists in the set, the new pair is not added. A hashset is used, allowing insertion of new coordinates in constant time given an appropriate hash function.

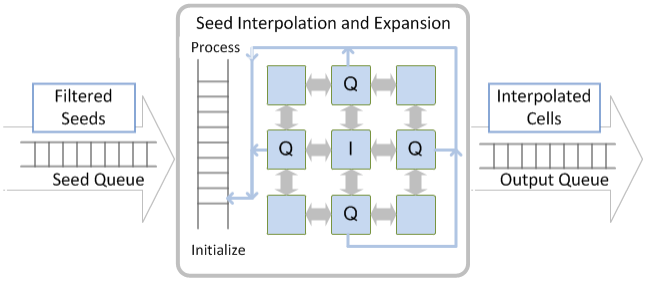

Figure 4 Region growing breadth first predicate evaluation

Seed Expansion: Seed expansion operators are candidates for operator cloning, as all seed expansion operators fetch seed cells from the same queue, but maintain their own internal cell candidate queues (see Figure 4). An index is shared between all parallel cell expansion operators; this index keeps track of whether cells have been tagged for expansion or not, avoiding duplicated efforts among different operator threads. Each individual seed expansion operator performs as follows:

A seed expansion operator fetches a seed cell from the seed queue, and grows an expanding region based on the seed. Region growing is continued as long as the predicate is satisfied. The region-growing algorithm is a breadth first search, but without the termination condition of finding a particular item. Instead, we added the requirement that the predicate is satisfied for a cell before neighbors are added to the queue. The breadth first algorithm interpolates the seed cell's value and if it satisfies the predicate, the cell is expanded. This means that four of its neighboring cells (north, east, south, and west) are added to its own "candidate cell" queue for this region. Before cells are added to the queue, the operator checks whether the cell has already been tagged for expansion in any of the other operator threads.

Next, the head of the candidate cell queue is fetched, interpolated and conditionally expanded, and so on. The algorithm terminates once the internal candidate cell list is empty, and then a new seed is fetched from the seed cell queue and the process continues.

4.2.2 Scanline

The Scanline algorithm is an alternative seed based region growth algorithm; it can be visualized as starting from a seed cell to traversing a line in a raster to find relevant neighbors. In image processing, each cell's values are known. This is not true for our filter based predicate evaluation process, in which cells have to be interpolated *before* they can be tested.

Seed Selection: Identical to Breadth First (see 4.2.1)

Seed Expansion: The basic idea of the Scanline is as follows: Fetch a seed from the seed queue. Then, start by filling the current 'scanline' in which the seed is located, i.e. proceed along the cells to the right of the seed in the current row: interpolate each cell and test if it satisfies the predicate. If so, visit the next cell to the right. Once a cell with a value that does not meet the predicate is encountered, the algorithm jumps to one of the lines that are vertically adjacent (upper or lower) and continues. Again, the algorithm tests all cells until it reaches the other side of the row, and then again jumps. This is repeated until the connected region is fully filled. Compared to the basic Breadth First algorithm, which searches in four directions, we mark the cells satisfying the threshold by traversing line sections.

4.3 Alternative 2: Tile expansion

If a grid is very large or has a high resolution, the identification of cells that are potentially elements of the predicate result *individually* through search can lead to substantial computational effort. In order to limit the steps of region growing, we explored using larger tiles as a basic test object to expand. A tile is a set of cells with equal width and height (e.g. 2x2 or 4x4 cells). The entire observation region is partitioned into equal-sized tiles, and if a candidate tuple is found in a tile, all cells of this tile are selected as candidates for ST-interpolation. Overall the purpose of tile expansion is to locate all (and likely more) grid cells that, after interpolation, will be an element of the predicate result. Tiles expansion is a rough, but fast estimate of such cells.

Tile expansion consists of an initial *seed tile selection* phase, and a *seed expansion* phase. Seed tiles are tiles with cells that contain seed tuples; expanded tiles are all neighboring tiles of seed tiles. The motivation for *tile expansion* is as follows: assume a seed tuple t_s exists in a corner cell c_i of tile T_j. It is possible that t_s will influence the interpolated value of grid cell c_n, which is a neighbor cell of c_i but located outside of tile T_j. Cell c_n might not have seed tuples itself. Therefore, seed tiles need to be expanded. Another consideration is *tile size*. If the tile size is large, the percentage of irrelevant cells being interpolated likely is larger. Choosing a small tiles size, however, will increase the memory and computational cost for the algorithm. Further, a dependency between tile size and chosen interpolation radius for the ST-interpolation method exists. If the interpolation radius r=16 (i.e. 16 cells in distance from center cell), choosing a tile size that is a multiple of 2 is advantageous (e.g. 2, 4, 8 or 16). The steps of tile expansion are as follows:

Seed selection: During the filtering of incoming streams, tuples are tested based on their value, and a passing tuple's (x,y) coordinate values are converted to *tile coordinates*. A new coordinate pair containing the tile coordinates is added to the set of *seed tiles*. If the tile already exists in the set, the new pair is not added. As with the breadth-first region-growing approach, a hashmap is used to index tiles and prevent duplicates. Tile identification is run as a single thread, and runs through all candidate tuples for a query window. The result is a set of unique tiles.

Expansion of seed tiles: Next, a new set of expanded tiles is created. The expansion operator iterates through the set of seed tiles and each seed tile is added to the set of expanded tiles along with all tiles within a square defined by the interpolation radius around the perimeter of the tile. The seed tile expansion guarantees that all cells that potentially have an interpolated value satisfying the predicate, since they are neighboring tiles within the interpolation radius of seed cells, will be interpolated. However, this 'overshooting' of cell selection comes at the cost of queueing cells that likely will not satisfy the predicate.

Conversion of expanded tiles to cells: The set of expanded tiles is converted into individual grid cells, which are queued for ST-IDW. In all cases the complexity for this is O(n*(r/s)^2) where n is the number of expanded tiles, r is the interpolation radius, and s is the number of tiles in a distance of r.

4.4 Discussion

Overall, the performance of the selected algorithms depends on the following components: indexing of raw tuples, filtering, seed selection, and seed expansion with interpolation (either by region growing or tile expansion), and interpolation. The cost for indexing raw tuples and filtering is O(d) (d, number of tuples). The cost for seed expansion in the region-growing approach has the following behavior: grid cells are arranged in a graph, in which each seed cell has four direct neighbors, which are visited. In the worst case, the entire observation region is part of the predicate result, and every edge in the graph has to be traversed. In Breadth First, all four neighbors are checked for each cell. In Scanline, often only a single neighbor is checked, but in both cases in the worst case all cells are interpolated. The time complexity for seed expansion using Breadth First is O(|E|), where the number of edges in the graph is |E|=2mn-m-n, m and n being the width and height of the grid. The space complexity is O(|V|), where |V|=mn is the number of cells in the grid. Overall, the cost is determined by the interpolation cost and grid size, and depends linearly on the number of tuples.

5. PHENOMENON-AWARE PREDICATE EVALUATION

We observe that predicate operators over dynamic phenomena that deliver predicate results continuously capture the incremental change of the phenomenon over time and space. For predicate evaluation, this indicates that we can exploit knowledge from a previous query window t_{i-1} during evaluation of the current window t_i and thus, limit the search for cells that are elements of the predicate result set. In the related work, this type of evaluation is named *incremental stream query evaluation*, i.e. the result of a new query window is built based on the result of the previous window, and new incoming tuples are individually added while outdated tuples are deducted, and the result is computed. This avoids re-computing the entire result from scratch.

5.1 Rationale

The phenomenon-aware predicate evaluation approach is based on *tiles* (Section 4.3). The motivation for tiles is to create a rough identification of the cells that are elements of the predicate operator. The observation region is partitioned into equal-sized tiles, and a tile 'mask' is created. Depicting the predicate result of window t_{i-1}, the result region(s) are contained in a set of tiles (see Figure 5, all blue tiles). We classify tiles into the following types:

Interior tiles: an interior tile contains only cells that fulfilled the predicate in t_{i-1}. Interior tiles contain cells that are most likely also part of the result of window t_i.

Boundary tiles: A boundary tile consists of both cells that fulfilled the predicate during t_{i-1} and cells that did not fulfill the predicate in the previous window. Boundary tiles capture areas that are likely to change from query window t_{i-1} to t_i.
Exterior tiles: exterior tiles contain only cells that did not fulfill the predicate in t_{i-1}. Exterior tiles might still be of the same types in window t_i, but new regions might emerge.
Based on the tile classification, a strategy for tile expansion is selected: interior tiles do not need to be expanded, boundary tiles are expanded in certain cases, and exterior tiles are only expanded if they contain new seeds that did not exist in t_{i-1}. In principle, the algorithm only considers tiles with 'change' potential through expansion and testing of the surrounding region, prunes areas that do not qualify, and re-interpolating regions that are still central to the predicate.

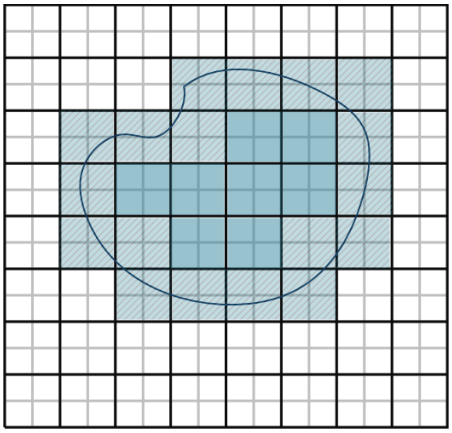

Figure 5 Tiles classification based on the t_{i-1} query window

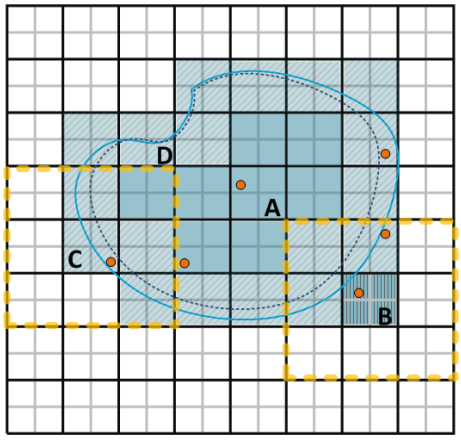

Figure 6 Tile classification and expansion based on new seeds

5.2 Classifying tiles

The phenomenon-aware predicate evaluation algorithm receives the following input: a) a queue of all interpolated cells of window t_{i-1}, including cells that are not part of the result set but were interpolated nevertheless, b) a tile mask in which each tile is marked as being in one of four states: No Data, Interior, Boundary or Exterior, and c) a queue of all seed tuples for t_i.
Tile classifier: Initially, all tiles are in the state "No Data". First, the operator consumes a queue element of its input queue, i.e. all interpolated cells from t_{i-1}. Based on a cell's location, the appropriate tile is identified, and based on the cell's value the tile is reclassified as *Interior* (if the cell's value fulfills the predicate), or *Exterior* (if the cell's value does not meet the predicate). If a cell arrives that does not meet the predicate but falls into a tile that has already been classified as Interior, the tile is reclassified as *Boundary* tile. After the cell queue is empty, all tiles are tagged Interior, Boundary, Exterior or No Data. No Data tiles are possible since the predicate result set is a subset of the entire region (see Figure 5).

5.3 Expansion of tiles types

In the next step, the queue of new seed tuples, which arrived during t_i is processed, and the tiles are reclassified (see Figure 6, red dots). If a seed is identified within an interior tile, the tile status does not change since all cells of the tiles will be interpolated in any case (Figure 6, tile A). If a new seed is located within a boundary tile, the boundary tile is marked for expansion (Figure 6, tile C). The new seed indicates that the predicate result region likely has grown within this tile, thus neighboring tiles should be considered. If a seed falls into an Exterior tile, the Exterior tile is marked for expansion and handled like an initial seed tile (expanding to neighboring tiles) (Figure 6, tile B). If a seed falls into a No Data tile, the tile is reclassified as Exterior, and marked for expansion.
At the end of this phase, all tiles are marked as *Interior*, *Boundary/NotExpand*, *Boundary/Expand*, *Exterior/Expand*, *Exterior* and *No Data*. At this time the *Exterior* and *No Data* tiles are pruned and not further considered. The cells of the *Interior* and *Boundary/NotExpand* tiles are inserted directly into the output queue for cell interpolation. *Boundary/Expand* and *Exterior/Expand* tiles are expanded first to their neighboring tiles; cells of all expanded tiles are then inserted into the output queue for the ST-interpolation operator.

6. PERFORMANCE EVALUATION

6.1 Experimental setup

Since sensor data streams of high density, both spatially and temporally, are not available (yet) for the system we envision, we simulated sensors moving along a dense (Cambridge) and a sparse street network (one third of Japan around the 2011 Fukushima nuclear incident). The discussed strategies were implemented as data stream components assuming a DSE environment.

6.1.1 Data sets

To generate high-density data streams, a simulation of sensor streams in NetLogo [20] was created. The movement of sensor nodes is constrained to links in a street network. We implemented two street networks; one is a large portion of Japan and the other is a small part of the Cambridge and Boston, MA area. As phenomenon, we use the measured and estimated radiation deposition levels in Japan after the Fukushima Daiichi nuclear disaster in March 2011. The predicted radiation levels were calculated in R using data from ZAMG [30] and SPEEDI [6]. The simulation loops through a sequence of snapshots of the event between the 1st and 5th day after the disaster in 15-minute increments. Each increment is represented as a 'tick' in Netlogo, and a query window is defined over 4 ticks. In a realistic system, we would expect each tick to correspond to 15 seconds or less. The simulated sensors on the street networks sample the phenomenon and write observations to a text file. The data sets were generated for a simulated sensor population of 256K that sampled at each tick.

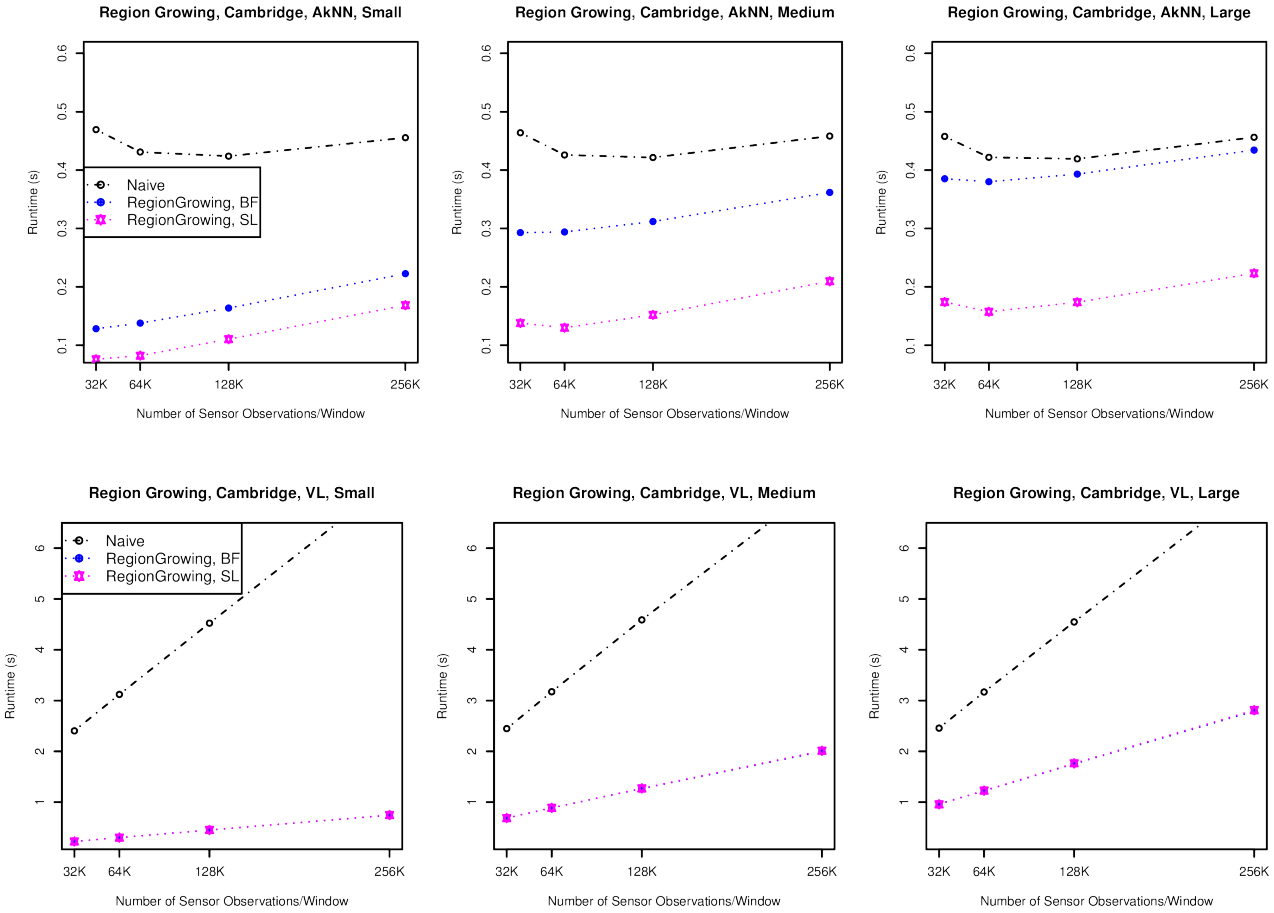

Figure 7 Region growing-based algorithms with akNN and Virtual List (VL)

6.1.2 Implementation und Runtime Environment

The algorithms are implemented in Java in a limited DSE environment; operators are connected via queues, and work in a pipelined fashion, but we do not consider any of the other DSE components (e.g. adaptive resource optimization, etc.). The experiments were run on a Mac Pro (Model MacPro6.1) with a 3.5 GHz Intel Xeon E5 (six physical processing cores and 12 virtual cores), 16 GB DDR3 ECC memory at 1867 MHz, and Mac OS X 10.9.4 (13E28) (64 bit), and Java 1.7.0_60 (64 bit). Tests were run with the heap size configured to an initial and maximum size of 4 GB in order to buffer the entire input stream in memory.

6.2 Parameters used based on previous work

For the following tests a sliding window with a range of 4 ticks (no slide, tumbling window) was selected. Tests were run with ten iterations of five consecutive windows over both the Cambridge and Japan data sets. The data set is composed of simulated sensor observations with 256K sensors, each of which has a 25% probability of updating for a particular tick (i.e. on average each sensor updates at one point in the window). A 512x512 grid was used. In previous work [28], we developed two methods that determine surrounding tuples that are needed to interpolate the value of a cell. The akNN algorithm dynamically increases the search radius, both in space and time, around the cell to be interpolated until it has found k samples. The alternate Virtual List (VL) approach uses a fixed spatial search radius *r*, and all samples in the search radius are used as input for interpolation. Based on previous results [28], we have established that the adaptive kNN based ST-IDW has better performance in all cases (uniform and skewed sampling distribution). Therefore, for most performance results we show only the akNN results, but some results for VL are included for comparison.

For akNN, parameter k was set to 4. The interpolation center is at the end of the query window and a single snapshot of the predicate result is created. The pipelined stream query execution was run as follows: one indexing thread, one seed selection and expansion thread, and eight interpolation threads. Timing was done using mutual exclusion between the indexing and seed expansion operators, where applicable, to give consistent timing results, although they normally would run in parallel. Total runtime was calculated as: Runtime=Slowest(AVG(index-time), AVG(seed-time)) + AVG(Slowest (IDW1... IDW8)).

We tested the all methods with predicate settings that returned different sized results. The predicate result size was quantified as the proportion of number of cells of predicate result compared to number of cells of entire observation region. We tested the following cases: *small* (10%), *medium* (28%), and *large* (38%).

6.3 Naïve vs. region growing algorithms

First, we compared the performance of the naïve method (Naïve) to the region growing algorithms, i.e. ***Breadth First*** (BF) and ***Scanline*** (SL) (Figure 7). In the naive method all cells of the entire observation region are interpolated and predicate evaluation was performed after interpolation. The region growing algorithms start by filtering tuples and expanding seed cells. We tested the naïve and region growing methods for determining the predicate result in combination with the akNN (top 3 graphs in Figure 7) and VL algorithms (lower 3 graphs) used for interpolation. Further, we tested three different predicate result sizes. The results for VL show that both BF and SL have almost identical performance as expected; they are significantly faster and scale much better with increasing number of observations compared to Naïve. The interesting observation is that SL performs significantly faster than BF when combined with the akNN algorithm (top 3 graphs). This is due to exploiting the high-speed caching as a linear traversal during interpolation results in most of

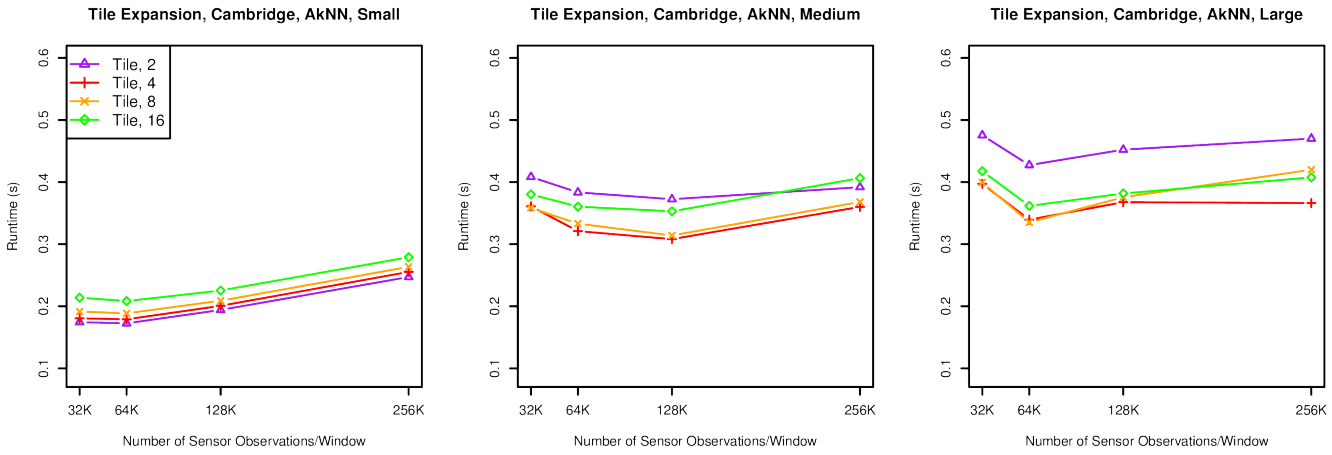

Figure 8 Tile-based expansion

the data needed to interpolate the current grid cells was already loaded into the cache hierarchy when interpolating the previous cell.

Overall, the region growing algorithms perform better than Naïve for akNN. We also observe that the runtime initially decreases with increasing number of observations since the search can be reduced to fewer cells in akNN. However, the performance decreased again once the set of observation reaches about 128K samples/window as expected. Comparing akNN and VL, the akNN version is significantly faster than VL (e.g., runtime between 0.14-0.2s for akNN/SL, medium result size compared to 0.8-2s for VL/SL, medium result size).

6.4 Tile expansion

We tested the performance behavior of the tile expansion approach. This approach is expected to behave more greedily and faster in identifying areas that are part of the predicate result instead of exploring the region cell by cell. Tile sizes tested were S_1=2x2, S_2=4x4, S_3=8x8, and S_4=16x16. The history-unaware tile expansion approach only uses the seeds from the current window while the history-aware, informed approach (Section 5) uses the information about the extent of the phenomenon gleaned from the last interpolation step (the comparison is discussed in Section 6.5). The results for tile size impact are shown in Figure 8. As expected, the uninformed tile expansion algorithm with tile size 2 performs best if the predicate result is small, and worst if the predicate result size reaches 38% of the overall region. Tile size 16 shows slower performance than other methods in most cases. Tile sizes 4 and 8 show good performance on average. A tile size of four is the best choice for consistently good performance for akNN for both Cambridge and Japan and all three phenomena sizes.

6.5 Comparison of all methods

Figure 9 shows a comparison of all tested methods: *Region growing* (Breadth First and Scanline), *tile expansion* (informed and uninformed), and *naïve*. For tile expansion methods, the graphs for the same tile size have the same color (e.g. tile size 16 is green for both informed and uninformed), the informed tiles methods are dashed lines, while the uninformed tiles method results are solid lines. We show results for the Cambridge, MA and Japan areas. The difference between the datasets is that Cambridge is sampled via a dense road network, which results in spatially uniform sampling, while the Japan region has sampling skewed towards the a sparse road network.

For the VL tests (lower 2 graphs of Figure 9), results confirm that the Naïve method has the longest runtime, while the region growing methods has the best performance compared to the tile based and the naïve approaches, both over Cambridge and Japan. The tile-based approach behaves almost identically for tile sizes 2, 4, and 8, with a large tile size (16) decidedly slower than smaller tiles sizes.

For akNN (top two graphs of Figure 9), we can see that SL outperforms all other methods by a large margin. The results also show that the informed, phenomenon-aware tile based approach is consistently more efficient than the uninformed tile approach for tiles size 2, 4 and 8, but worse in performance for tile size 16. We conclude that incremental query evaluation for predicates over spatio-temporal fields is beneficial. However, using Scanline in combination with akNN was the best option given the characteristics of the test cases.

7. RELATED WORK

With regard to spatio-temporal applications, DSEs have mostly been used for handling sensor data streams that represent moving point-based objects like vehicles or people in real-time or RFID scanning [16–18, 26]. Our work of developing a DSE framework to monitor phenomena that are continuous over both space and time in real-time is currently unique [22, 28]. When monitoring moving objects, location is the measurand and object trajectories are interpolated over space and time. Predicates are posed over those trajectories, and are categorized into range queries (e.g. “which cars crossed region A in the last 10 min”), kNN queries (e.g. “which are the k nearest neighbors of each car during the last window”), or aggregate kNN queries (e.g. “which are the cars with minimum aggregate distance from object A during the last window”) [8]. When monitoring fields, predicates over the thematic values are of significant interest (e.g. “find all regions where the temperature is above 110°F in the last window”, “find all regions where the temperature increases by more than 3°F over each consecutive window”). Typically, fields are analyzed with a large set of other spatial analysis operators [7] but spatio-*temporal* analysis operators are less common today. Currently, pipelined stream based implementations [11] of these operators necessary for efficient execution with DSEs are not yet available. Our work in this paper is a first investigation into the foundation of evaluating predicates over ST-continuous phenomena with DSEs.

Compared to moving objects, which can be represented as points distributed over space, fields have a spatially *continuous* aspect, and generating a continuous representation based on potentially moving sensor stations with point-based samples is a crucial first step. This characteristic also has a significant impact on predicate evaluation.

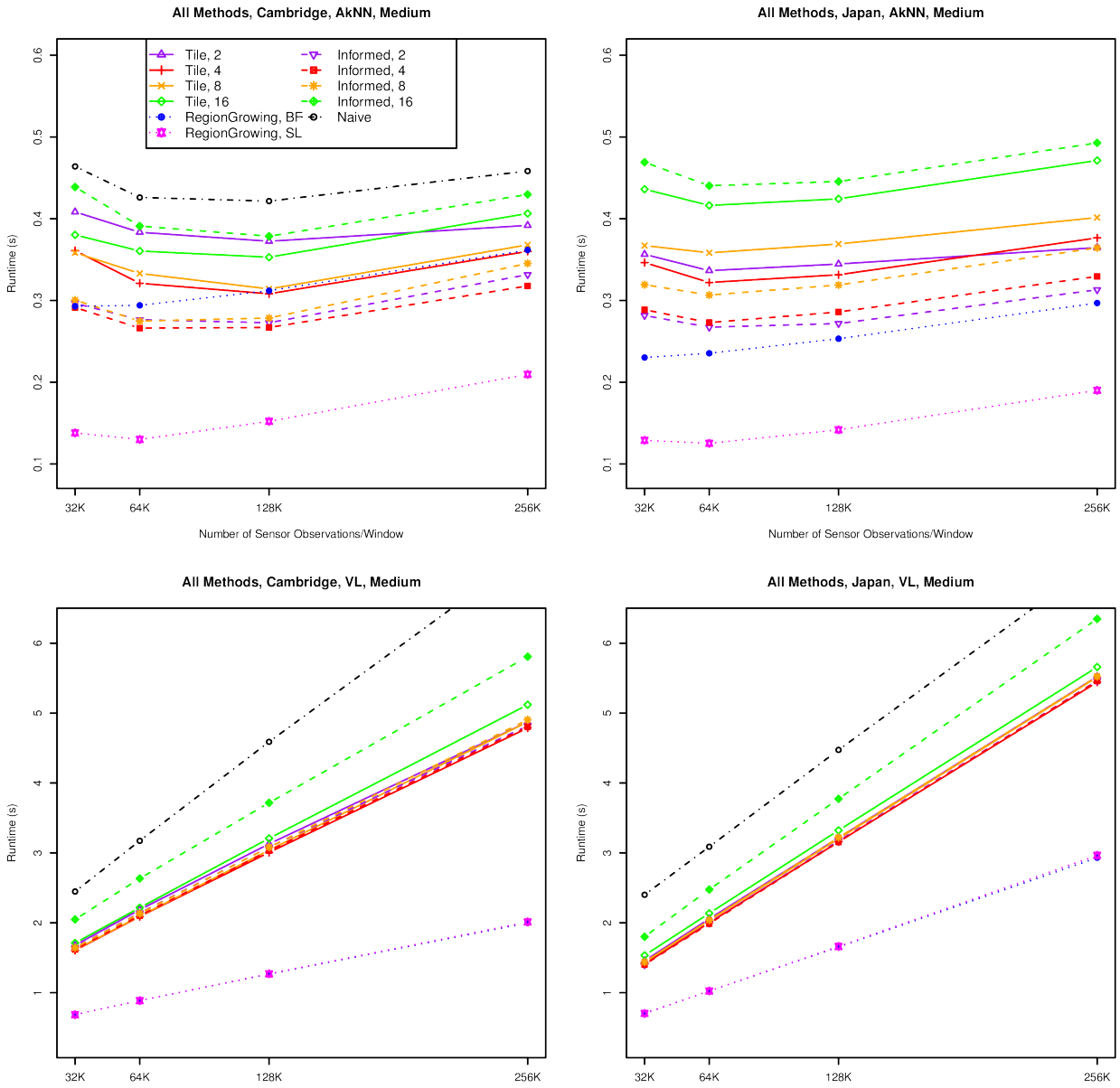

Figure 9 Summary

Important investigations and contributions in the area of pipelined, stream-based query operators and plans have been made on the topic of spatio-temporal predicates over moving objects, notably in Place [17], SINA [16], Nile [10], CAPE [26] and others. A summary can be found in [8]. The key characteristic for spatio-temporal stream queries is that often *in-memory* spatial index structures are used to index incoming tuples/window to evaluate spatial queries faster. Building pipelined query operators and plans that can be parallelized and evaluated incrementally has been one of the major challenges [8]. In moving object queries, the trajectory information of objects is of interest and has to be maintained for queries across query windows, but in continuous phenomena queries the challenge is to continuously create updated presentations over the entire phenomenon or a large enough portion as a basis to evaluate predicates correctly. The presented work is a first investigation into the foundation of evaluating predicates over continuous phenomena with DSEs.

8. CONCLUSIONS AND FUTURE WORK

Technological advances have created an unprecedented availability of inexpensive sensors able to stream environmental data in real-time for which we still seek appropriate data management technology that can keep up with this onslaught of sampling in previously unavailable spatial and temporal density. In previous work we have shown that DSEs can be extended to generate smooth representations of continuous spatio-temporal fields sampled by up to 250K sensors on-the-fly in near real-time, creating a new representation every second. In this paper we have investigated a spatio-temporal stream operator framework that efficiently executes predicate operators over spatio-temporal fields. We introduced a definition of predicates over dynamic fields, analyzed requirements for stream query evaluation and presented several pipelined stream based query operators algorithms. The work is based on the assumption that it is more efficient to find 'seeds' of regions that are part of the predicate result and expand them into the complete predicate result regions instead of interpolating the entire continuous phenomenon first, and filtering all cells based on the predicate condition. We investigated different seed expansion algorithms (*Breadth First* and *Scanline region growing*, and *tile expansion*) as well as exploring the impact of using the *knowledge of the previous window query* result. Our analysis and performance results show that both region growing algorithms perform best for all data set sizes and characteristics; tile-based approaches are efficient for tiles sizes 4x4 and 8x8. History-aware tile expansion performs better if the phenomenon changes slowly (as expected). Future work will include investigating adaptive query evaluation using

the different algorithms based on the changing phenomenon characteristics.

Acknowledgements

The authors would like to thank Christopher Dorr for his valuable input to this paper.

REFERENCES

[1] Ali, M. et al. 2010. Real-time spatio-temporal analytics using Microsoft StreamInsight. *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems - GIS '10* (New York, New York, USA, 2010), 542–543.

[2] Ali, M. et al. 2011. The extensibility framework in Microsoft StreamInsight. *2011 IEEE 27th International Conference on Data Engineering* (Apr. 2011), 1242–1253.

[3] Arasu, A. et al. 2005. The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal*. 15, 2 (Jul. 2005), 121–142.

[4] Biem, A. et al. 2010. IBM InfoSphere Streams for Scalable, Real-Time, Intelligent Transportation Services. *SIGMOD '10: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data* (Indianapolis, IN, 2010), 1093–1103.

[5] Campbell, A.T. et al. 2008. The Rise of People-Centric Sensing. *IEEE Internet Computing* (Jul. 2008), 30–39.

[6] Chino, M. et al. 1993. SPEEDI and WSPEEDI: Japanese Emergency Response Systems to Predict Radiological Impacts in Local and Workplace Areas due to a Nuclear Accident. *Oxford JournalsMathematics & Physical Sciences Radiation Protection Dosimetry*. 50, 2 (1993), 145–152.

[7] Chrisman, N. 1999. A transformational approach to GIS operations. *International Journal of Geographical Information Science*. 13, 7 (1999).

[8] Elmongui, H.G. et al. 2006. Challenges in Spatio-temporal Stream Query Optimization. *MobiDE '06 Proceedings of the 5th ACM international workshop on Data engineering for wireless and mobile access* (2006), 27–34.

[9] Galton, A. and Worboys, M. 2005. Processes and Events in Dynamic Geo-Networks. *GeoSpatial Semantics*. M.A. Rodríguez et al., eds. Springer LNCS 3799. 45–59.

[10] Hammad, M.A. et al. 2004. Nile: A query processing engine for data streams. *Proceedings of 20th International Conference on* (2004), 851.

[11] Kazemitabar, S. et al. 2010. Geospatial stream query processing using Microsoft SQL Server StreamInsight. *Proceedings of the VLDB Endowment*. 3, 1-2 (2010), 1537–1540.

[12] Kemp, K.K. 1996. Fields as a framework for integrating GIS and environmental process models. Part 1: Representing spatial continuity. *Transactions in GIS*. 1, 3 (1996), pp 219–234.

[13] Kemp, K.K. et al. 1998. Towards an ontology of fields. *Geocomputation* (1998), 1–6.

[14] Miller, J. et al. 2011. An Extensibility Approach for Spatio-temporal Stream Processing using Microsoft StreamInsight. *SSTD'11: Proceedings of the 12th international conference on Advances in spatial and temporal databases* (2011), 496–501.

[15] Mitas, L. and Mitasova, H. 1999. Spatial interpolation. *Geographical Information Systems: Principles, Techniques, Management and Applications*. P. Longley et al., eds. Wiley. 481–492.

[16] Mokbel, M. et al. 2004. SINA: Scalable incremental processing of continuous queries in spatio-temporal databases. *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data* (2004), 623–634.

[17] Mokbel, M.F. et al. 2005. Continuous Query Processing of Spatio-Temporal Data Streams in PLACE. *GeoInformatica*. 9, 4 (Nov. 2005), 343–365.

[18] Mokbel, M.F. and Aref, W.G. 2007. SOLE: scalable on-line execution of continuous queries on spatio-temporal data streams. *The VLDB Journal*. 17, 5 (Apr. 2007), 971–995.

[19] Murty, R.N. et al. 2008. CitySense: An Urban-Scale Wireless Sensor Network and Testbed. *2008 IEEE Conference on Technologies for Homeland Security*. (May 2008), 583–588.

[20] Netlogo Home Page: 2014. .

[21] Ng, K. et al. 1999. Dynamic Query Re-Optimization. *SSDBM '99 Proceedings of the 11th International Conference on Scientific and Statistical Database Management* (1999).

[22] Nittel, S. et al. 2012. Real-time spatial interpolation of continuous phenomena using mobile sensor data streams. *Proceedings of the 20th International Conference SIGSPATIAL '12* (New York, New York, USA, 2012), 530.

[23] Oracle, A. and Paper, W. 2008. Oracle Complex Event Processing Performance Oracle Complex Event Processing Performance. November (2008).

[24] Paulos, E. et al. 2008. Citizen science: Enabling participatory urbanism. *Urban Informatics: community Integration and Implementation*. 1–16.

[25] Resch, B. et al. 2009. Real-Time Geo-awareness Sensor Data Integration for Environmental Monitoring in the City. *2009 International Conference on Advanced Geographic Information Systems & Web Services* (Feb. 2009), 92–97.

[26] Rundensteiner, E.A. et al. 2005. CAPE: A constraint-aware adaptive stream processing engine. *Stream Data Management, Advanced in Database Systems*. N. Chaudhry et al., eds. Springer Berlin Heidelberg.

[27] StreamBase 2012. StreamSQL online documentation.

[28] Whittier, J.C. et al. 2013. Towards Window Stream Queries over Continuous Phenomena. *Conf Proc of 4th International Workshop on "Geostreaming", in conjunction with SIGSPATIAL* (Orlando, FL, 2013), 1–10.

[29] Witkowski, A. et al. 2007. Continuous queries in Oracle. *VLDB '07 Proceedings of the 33rd International Conference on Very Large Data Bases*. (2007), 1173–1184.

[30] Wotawa, G. and Skomorowski, P. 2012. Long-range transport of particulate radionuclides from the Fukushima NPP accident: sensitivity analysis for wet deposition. *EGU General Assembly 2012* (Vienna, Austria, 2012), 10494.