Summer 8-17-2018

# Towards an Efficient, Scalable Stream Query Operator Framework for Representing and Analyzing Continuous Fields

John Whittier
john.c.whittier@maine.edu

# TOWARDS AN EFFICIENT, SCALABLE STREAM QUERY OPERATOR FRAMEWORK FOR REPRESENTING AND ANALYZING CONTINUOUS FIELDS

By

John Chandler Whittier

B.S. University of Maine, 2010

A DISSERTATION

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

(in Spatial Information Science and Engineering)

The Graduate School

The University of Maine

August 2018

Advisory Committee:

Silvia Nittel, Associate Professor in Spatial Information Science and Engineering, Advisor

Kate Beard-Tisdale, Professor of Spatial Information Science and Engineering

Torsten Hahmann, Assistant Professor of Spatial Information Science and Engineering

James Fastook, Professor of Computer Science

Ali Abedi, Professor of Electrical & Computer Engineering

# TOWARDS AN EFFICIENT, SCALABLE STREAM QUERY OPERATOR FRAMEWORK FOR REPRESENTING AND ANALYZING CONTINUOUS FIELDS

By John Chandler Whittier

Dissertation Advisor: Dr. Silvia Nittel

An Abstract of the Dissertation Presented
in Partial Fulfillment of the Requirements for the
Degree of Doctor of Philosophy
(in Spatial Information Science and Engineering)
August 2018

Advancements in sensor technology have made it less expensive to deploy massive numbers of sensors to observe continuous geographic phenomena at high sample rates and stream live sensor observations. This fact has raised new challenges since sensor streams have pushed the limits of traditional geo-sensor data management technology. Data Stream Engines (DSEs) provide facilities for near real-time processing of streams, however, algorithms supporting representing and analyzing Spatio-Temporal (ST) phenomena are limited.

This dissertation investigates near real-time representation and analysis of continuous ST phenomena, observed by large numbers of mobile, asynchronously sampling sensors, using a DSE and proposes two novel stream query operator frameworks. First, the ST Interpolation Stream Query Operator Framework (STI-SQO framework) continuously transforms sensor streams into rasters using a novel set of stream query operators that perform ST-IDW interpolation. A key component of the STI-SQO framework is the 3D, main memory-based, ST Grid Index that enables high performance ST insertion and deletion of massive numbers of sensor observations through Isotropic Time Cell and Time

Block-based partitioning. The ST Grid Index facilitates fast ST search for samples using ST shell-based neighborhood search templates, namely the Cylindrical Shell Template and Nested Shell Template. Furthermore, the framework contains the stream-based ST-IDW algorithms ST Shell and ST a$k$-Shell for high performance, parallel grid cell interpolation. Secondly, the proposed ST Predicate Stream Query Operator Framework (STP-SQO framework) efficiently evaluates value predicates over ST streams of ST continuous phenomena. The framework contains several stream-based predicate evaluation algorithms, including Region-Growing, Tile-based, and Phenomenon-Aware algorithms, that target predicate evaluation to regions with seed points and minimize the number of raster cells that are interpolated when evaluating value predicates.

The performance of the proposed frameworks was assessed with regard to prediction accuracy of output results and runtime. The STI-SQO framework achieved a processing throughput of 250,000 observations in 2.5 s with a Normalized Root Mean Square Error under 0.19 using a $500 \times 500$ grid. The STP-SQO framework processed over 250,000 observations in under 0.25 s for predicate results covering less than 40% of the observation area, and the Scan Line Region Growing algorithm was consistently the fastest algorithm tested.

## Contents

## LIST OF FIGURES

## LIST OF QUERIES

# CHAPTER 1
# INTRODUCTION

## 1.1 Real-Time Sensing in Sensor-Rich Environments

Technology advancements in wireless data networks, microelectronics, and mobile devices have made it easier and less expensive to deploy and receive live data from sensors. Sensors are able to function independently or connect to other devices in order to upload observations, and they are no longer constrained to stationary locations. These technology developments enable dense deployments of networked sensors, developed and installed by municipalities [144], researchers [97, 116, 177], and concerned citizen who engage in *participatory sensing* [2, 25, 27, 60, 115, 142], where individuals operate environmental sensors that live stream data to be shared and analyzed as part of a larger, collective data set.

Sensor platforms used in such deployments are often comprised of stand-alone devices assembled using inexpensive microcontrollers with open source hardware and software such as Arduino [15], BeagleBone [20], Intel Galileo [73], Libelium Waspmote [93], and Raspberry Pi [137] development boards. With these development boards it is easy to combine hardware and software components for a particular task. Another type of commonly used sensing device is a smartphone, which can connect to external environmental sensors [21, 102, 161, 167] or use built-in sensors including temperature, humidity, barometric pressure [143], air quality including fine particulate matter (e.g., $PM_{2.5}$) and Volatile Organic Compoundss (VOCs) [24, 125], and radiation levels [147].

Many early examples of deploying stationary and mobile sensors for monitoring different environmental phenomena exist today [67, 97, 116, 139, 144, 142, 177]. For instance, ocean observing buoys such as the NOAA National Buoy Data Center [118], the NERACOOS system [120], or the Argo network [139] consist of large numbers of live

streaming sensor buoys. Other deployments include sensor networks on municipal infrastructures such as light poles (e.g., CitySense [116], SmartSantander [144], Cambridge, UK [97]), or on buses or private vehicles [142, 144]. Recent applications feature sensors deployed in forests to measure wildfires (RISER [177], [44]), wearable sensors to measure air quality [64, 67], and sensor networks to monitor earthquake activity [45, 72, 79].

This type of technology infrastructure with sensors directly connected to the Internet enables the collection of observations from potentially hundreds of thousands of sensors deployed over a geographic area, such as a large city, at a high sample rate. It is now possible to observe continuous, environmental phenomena that are seamlessly distributed across a geographic region and exhibit continuous change over space and time in near real-time. Examples of such continuous phenomena are airborne particulate matter, chemical particles in the air or water, rainfall, or the dispersion and decay of radioactive particles.

Near real-time analysis of continuous, Spatio-Temporal (ST) phenomena observed by sensors is required to facilitate rapid reaction to events. This is particularly relevant in the context of emergency management, where near real-time performance becomes critical to minimize impact on citizens. In the event of a nuclear or chemical plume disaster, citizens need timely access to information about the areas impacted by the disaster in order to reduce personal exposure and to identify evacuation routes.

### 1.1.1 Challenges

There are several challenges associated with the availability of massive sensor streams that enable us to observe spatio-temporally continuous phenomena in near real-time based on near live sensing and data delivery.

- **Mapping Massive Streams of Discrete Observations to ST Continuous Representations of Phenomena, in Near Real-Time**

  Each geosensor observation is a discrete observation, at an instant in time and point in space, of the spatially continuous phenomena that is continuously evolving over time. Aref [16] argues "that in sensor networks, processing should be at the phenomenon-level and not at the data signal level." In order to analyze the *phenomenon* and capture its continuity over space and time, the discrete observations need to be combined using the available data to predict the phenomenon for those locations that have not been sampled. A continuous representation is needed to visualize the phenomenon and necessary as a basis for analysis so that meaningful information can be extracted and communicated to users. This information includes which regions have hazardous radiation levels, which evacuation route will minimize personal exposure, or the calculation of a personal exposure given a person's trajectory [115].

  Representations of continuous phenomena based on discrete observation points are created by applying a spatial interpolation function to the point samples to calculate a value for the non-sampled points. The spatial interpolation functions are computationally expensive and today's implementations do not scale to keep up with the high velocity of arriving observations and demands of producing continuous representations in near real-time. In our context, we define near real-time as being able to generate a new representation every 1-3 seconds.

  Urban sensor networks can consist of thousands to millions of deployed sensors. Each of those sensors can produce an observation every few seconds. For such a sensor network deployment with one million sensors in which sensors update at roughly every 1-10 seconds, the stream could have an average data rate of 100,000 to 1 million observations per second. This presents several data management challenges: For a stream of sensor observations to be continuously processed in near real-time,

the system needs to be able to process data at the rate new observations arrive. This could mean processing 500,000 observations tuples/s, indefinitely. The processing needs to transform the individual sensor observations into a continuous representation of the phenomenon.

- **Continuous Phenomena Vary Dynamically over Space and Time**
  Characteristically, continuous phenomena are continuous both over physical space and also over time as they change, slowly or rapidly depending on the phenomenon. Sensors that are able to observe and deliver high frequency updates allow us for the first time to capture both types of change in near real-time. However, sensor samples are only taken at discrete locations and times while for analysis a continuous representation is necessary, which must be able to capture both the spatial and temporal continuity of the phenomenon. While creating spatially continuous representations at temporal snapshots with large temporal gaps is state of the art today, the estimation and representation of ST continuity is still largely unexplored. ST interpolation methods are necessary that estimate the behavior of the phenomenon as correctly as possible. Furthermore, they also need to scale their computational capacity to high-throughput sensor data streams.

- **Analyzing ST Continuous Phenomena based on Massive Sensor Data Streams in Near Real-Time**
  Automated analysis is an important aspect of continuously monitoring real-world processes and ST continuous phenomena. Such analysis, or queries, can find relevant trends and identify and alert to events, while irrelevant events can be filtered out and discarded. For users, it is convenient if analysis needs can be expressed as queries that can be easily configured and reused. For instance, queries could analyze the streams to detect areas where measurements are above a certain threshold, applicable for detecting areas with hazardous air pollution or radiation levels and to detect new

spot fires around a wildfire. First, appropriate query operators for ST Continuous Phenomena need to be identified. Secondly, methods need to be developed to execute such ST analysis queries *efficiently* over vast sensor data streams that only present discrete samples in arbitrary spatial order. Furthermore, such queries need to return *accurate* results for both interpolated values and detected areas satisfying a threshold. The point samples do not capture the continuity of the phenomena, and selecting the point samples that satisfy a query (e.g., samples that are above a minimum threshold) is different from selecting the portion of the phenomenon that satisfies the query. While areas near samples that satisfy the threshold are likely to also satisfy the threshold, samples not satisfying the threshold are also important in order to determine the limits on the area satisfied by the query. Fourth, since observations arrive continuously as a stream, analysis must be performed incrementally as the data arrive rather than waiting for all data to arrive. Sensor observations arrive at a high velocity, so processing must be fast in order to keep up with the arriving data.

## 1.2 Current State of the Art

As previously shown [156, 162, 90, 77], the high velocity of sensor observation production from very large sensor deployments exceeds the capabilities of traditional data management and query support for representing and analyzing continuous phenomena such as Geographic Information System (GIS) software [169] or a Spatial Database Management System (SDBMS) [148].

### 1.2.1 Traditional GIS and Database Management Systems

Geographic Information System (GIS) [22, 169] software packages contain a wide variety of support to represent and analyze spatial data. GIS include, among others, tools for working with point sensor observations, and converting point observations into different phenomenon representations; they also support functions for filtering point data as well as

phenomena by spatial and value conditions. However, GIS software is not built today to automatically handle data that arrive in a stream; instead, GISs are implemented to process data stored in files (e.g., ArcGIS [41]) or in-memory data frames (e.g., R [135] using the `gstat` package [128]) in which all necessary data is stored and available before an operation is started. Furthermore, GIS relying on a Spatial Database Management System (SDBMS) cannot handle the throughput requirements of processing point observations, producing representations, and performing analysis in near real-time for massive streams of sensor observations because the libraries were not designed to work with streaming data [162, 77].

Available with GIS software are spatial interpolation methods [84, 85, 103] that transform discrete point samples into a spatially continuous representation of a phenomenon. Different representations for continuous phenomenon are available [22, 35], including rasters, tessellations (e.g., TIN [131] and Voronoi diagrams [18, 23]), or contour lines (isoline framework [35]), and different methods exist to create such representations. In this thesis, we focus on raster representations for continuous phenomena since they are widely used and compatible with other representations such as satellite images. Two of the most common spatial interpolations methods to generate rasters are Inverse Distance Weighting (IDW) [149] and Kriging [80]. Kriging is a geo-statistical interpolation method with a high computational complexity ($O(n^4)$) compared to non-geostatistical methods such as IDW [149] ($O(n)$). When sensors are stationary, the performance can be further improved by reusing distance weights and covariance matrices when interpolating using new values [68]. The accuracy of an interpolated raster can be assessed by calculating the Root Mean Square Error (RMSE) between the interpolated raster and known values.

One way of capturing the change of a ST phenomena over time is a temporal sequence of *spatial* snapshots, called the snapshot model [17]. The snapshot model represents space as being continuous, as each spatial snapshot is continuous over space, but these snapshots are only at discrete time instants that may be regularly [54] or irregularly spaced. A

well-known representation of ST continuity using raster snapshots was provided by the space-time cube [54, 99].

SDBMS (e.g., Oracle Spatial and Graph [126], Microsoft SQL Server [100], PostGIS [133] extension for PostgreSQL), are designed to manage and query large spatial data sets conveniently. To do so they provide a data model language to describe the structure of spatial data objects and a query language to express declarative analysis queries over the spatial data. Today, SDBMSs are best suited for spatial objects, such as land parcels, roads, geographical features, and utility systems, which are represented as vector geometries and are updated infrequently. SDBMSs are built as extension to traditional, general purpose Database Management Systems (DBMSs), and, as is also required for DBMSs, data must always first be written to disk before they can be queried or analyzed. This requirement limits the velocity of a stream that DBMSs and SDBMSs are able to process. Comparing tuple processing throughput between a Data Stream Engine (DSE) and DBMS, [155] showed that on the same machine a DSE was able to process 160,000 tuples/s while a DBMS was only able to process 900 tuples/s, over two orders of magnitude faster. This means that DSEs are much better suited than SDBMSs for near real-time processing of high velocity data streams from massive sensor deployments.

### 1.2.2 Data Stream Engines

Since mid-2000, DSEs were designed and implemented as an alternative data management system to DBMS to provide high-throughput query processing for data streams [13, 14, 29, 31, 66, 109, 140]. Streams are buffered in main memory only and immediately processed, eliminating the disk bottleneck of a DBMSs and enable DSEs to achieve significantly higher processing throughput. Additionally, DSEs provide support for *continuous queries* [158, 33], which are queries over data streams that are continuously evaluated as new data arrive. A stream query can be applied to an entire stream that has arrived since the time the query was posed, or it is run over the latest portion of the

stream, a so-called *window* [14]. A window is defined by its *length* and *slide*, both specified in terms of either tuple count or time interval, which dictate how the window advances over the data stream as new data arrive. Data stream queries are translated into a query plan and executed by DSE using scalable, one-pass [117] stream operators, which can be parallelized and optimized [49].

### 1.2.3 Data Stream Engine Support for Continuous Phenomena

While DSEs are promising data management technology for processing continuous queries over high-throughput data streams in near real-time, they only provide limited support for the representation and querying of ST phenomena today. Many of the DSE extensions have focused on support for location update streams of moving objects with limited query operators and operator implementations [65, 105, 106, 109, 119]. Also, initial research has been done to extend DSEs for monitoring ST continuous phenomenon and generating continuous spatial representations from a stream of observations [95, 96, 123, 177, 176]. These extensions were initiated by our previous work [123, 92] where we extended DSE to generate raster representations as spatial snapshots and enabled a throughput of 250,000 sensor observations in under 3 seconds producing a raster with a resolution of $512 \times 512$ for sensors distributed on a street network. [176, 177, 95, 96] explored extending DSE to preform interpolation using Kriging. [176, 177] implemented spatial Kriging and [95, 96] ST Kriging for a DSE. The work has confirmed that Kriging does not scale to large numbers of sensors, with Kriging taking 2 seconds for processing 20 tuples using a $50 \times 50$ raster grid [176] and 30 seconds for 10 tuples using a $150 \times 150$ grid [95, 96], respectively. This is far below the expected arrival rate of 100,000 tuples/s.

### 1.3 Problem Statement

DSEs are a tool capable to provide the throughput needs for querying and analyzing phenomena in near real-time based on massive sensor data streams, but current support for querying spatio-temporally continuous phenomenon representation and analysis is limited. Most existing research efforts in using DSEs to query ST continuous phenomenon do not provide the throughput for processing a high number of point sensor observations in near real-time [95, 96, 176, 177]. For instance, the throughput of [95, 96, 176, 177] is limited to around 30 tuples/s to generate a new raster representation using adaptations of the Kriging [80] interpolation method, far from the requirement of being able to process 100,000 tuples/s.

Our previous work [123] focused on implementing a stream operator framework for IDW. This approach provided a significantly higher throughput, i.e., transforming 250,000 sensor observations into a $512 \times 512$ raster in under 2 seconds. However, this approach has several limitations. It assumes that all sensors update synchronously, once per window, and a purely spatial interpolation method is sufficient. However, these assumptions are often not realistic in the real-world since synchronizing the update behavior of around 1 million potential independent sensors is impractical.

Therefore, the following problems need to be addressed:

- **Near Real-Time Continuous Query Operators to Transform Asynchronous Sensor Data Streams into Representations of ST Phenomenon**

  In order to capture the ST continuity of a phenomenon, DSE query support is needed that can transform a massive stream of continuously arriving sensor observations into an accurate representation of the ST continuous phenomena in near real-time. Sensor deployments in large metropolitan environments, can contain sensors installation on public transportation vehicles and municipal infrastructure, e.g. light poles or traffic lights, which easily scale to millions of sensors. The streaming rates depend on the phenomenon that is observed and its rate of change; thus, a sensor network with

500,000 sensors with each sensor updating every 5 seconds can lead to 100,000 tuples/s. Thus, the system must be able to handle streaming data from sensors that are potentially mobile and sample asynchronously and be able to process 100,000 tuples/s.

- **Near Real-Time Continuous Query Operators for Simple Analysis Queries over Continuous ST Phenomena**

  DSE query operators for queries over ST phenomena are required. Today, query operators other than transforming point observation streams into continuous representations are not available. To support an intuitive user interface, operators should be based on the high-level abstraction of a continuous phenomenon instead of operators over individual observation streams. While [90] formally defined the signature of a set of query operators for ST continuous phenomena, which are based on relational algebra and include predicate, projection, union, and other operators, high-throughput, real-time algorithms for such query operators are not yet available today. These operators need to support streaming sensor observations, sensors that are asynchronous or mobile, high-throughput processing, and the ability to transform and analyze phenomena from discrete observations into a ST continuous representation in near real-time.

## 1.4 Research Objective and Contributions

The hypothesis of this dissertation is that using a stream query operator framework for a DSE to accomplish ST interpolation and predicate evaluation evaluation is an efficient approach for *representing* and *analyzing* continuous ST phenomenon, observed via massive numbers of asynchronous, mobile sensor streams if the framework employs an ST anisotropy-aware ST grid index with a shell-based search algorithm, combined with ST interpolation stream operators using an ST Adaptive $k$-Shell (ST a$k$-Shell)-based ST IDW interpolation algorithm, and predicate evaluation stream operators that localize predicate

evaluation to the areas that satisfy the predicate. Such a framework performs stream query evaluation in near real-time while achieving high accuracy of query results.

Specifically, such a DSE framework running on a on a single CPU socket workstation class machine should achieve a processing throughput target of 100,000 observation tuples/s, and generate a continuous representation in near real-time, repeatedly, every 1-3 seconds. Further, while achieving near real-time throughput, accuracy of interpolated values (assessed using Normalized Root Mean Square Error (NRMSE)) should not decrease compared to standard, non-real-time spatio-temporal interpolation. Similarly, the predicate result regions produced from real-time predicate evaluation over continuous representations should be accurate (evaluated using Jaccard index [74]).

### 1.4.1 Research Contributions

The contributions of this dissertations focus on a novel, scalable stream query operator framework that can be integrated into a DSE. The introduced stream query operator framework enables the high-throughput transformation of massive sensor data streams of point observation into representations of continuous ST phenomena. Additionally, the stream query operator framework supports the evaluation of high-level predicate queries posed over such continuous ST phenomena in near real-time. In detail, the following contributions are made:

- **Real-time Stream Query Operator Framework**

  We introduce the ST Interpolation Stream Query Operator Framework (STI-SQO framework) that transforms a high-throughput stream of sensor observations into continuous representations in near real-time. The transformation of streaming ST point clouds into spatial rasters is performed as a stream query that is evaluated via a set of novel stream query operators. The operators are a novel one-pass stream algorithm for IDW-based ST interpolation. The stream operators can be cloned and parallelized during run-time to reduce the computational bottlenecks of the

algorithm. The computational bottlenecks of the stream query framework are the identification of the subset of sample points, determined for each raster cell, to predict the value of each cell point and the calculation of each predicted value.

- **Flexible Query Result Presentations for Stream Queries over ST Continuous Phenomena**

  A stream query window captures an ST point cloud over a temporal interval. How should the ST continuous phenomenon be presented for the window? We introduce the concept of the *Interpolation Center (IC)* as a user-defined timestamp within each query window at which a ST snapshot of the continuous phenomenon is constructed using ST interpolation. The snapshot represents the predicted state of the phenomenon at that instant based on the observations from the entire window that are in spatial and temporal proximity to each prediction location. Thus, the user can control the time instant within each window. Further, we introduce several novel types of stream query result representations for ST continuous phenomena, that are either a single summary *snapshot* at the IC or a *window movie* composed of multiple snapshots showing an animation of the ST phenomena over a window.

- **ST Grid Index with Isotropic Time Cells**

  We introduce an in-memory ST grid index with Isotropic Time Cells to enable efficient selection of observations within a ST neighborhood around each raster cell specified using a combined ST distance. An ST anisotropy ratio is used to partition space-time into Isotropic Time Cells by adjusting the temporal length of ST index grid cells relative to the spatial size of a raster cell. The isotropic time cells are centered about the IC, enabling the ability to search, emanating from the center of each raster cell, for proximate observations in space and time. Additionally, the ST grid index is designed to be efficiently reused between windows, able to quickly identify the outdated portion of the index and clear tuples so that portion may be

refilled with the newest data added to the window. This results in each tuple being inserted into the index once, saving both computation time and memory usage.

- **ST Interpolation Algorithms**

  For the stream query operator framework, two Spatio-Temporal Inverse Distance Weighting (ST-IDW) operator algorithms for near real-time ST interpolation with a throughput of over 100,000 tuples/s are introduced. The ST Shell Approach (ST Shell) algorithm uses *all* observations within a spatio-temporal shell around each cell in the output grid and within the temporal bounds of the window. In the ST a$k$-Shell method, the observations used for a particular predicated cell are determined using an iteratively expanding ST search for a lower bound of $k$ points within a maximum ST radius $r$. The ST-IDW algorithms support any arrangement of sensor observations in a ST point cloud, allowing both asynchronous and mobile sensors. The reusable *Cylindrical Shell Template* and *Nested Shell Template*, for ST Shell and ST a$k$-Shell, respectively, are introduced as templates for searching the index for nearby tuples to each cell interpolated. We introduce an ST anisotropy ratio to merge spatial distance and temporal difference between and into a combined ST distance weight.

- **Data Skew Adaptive kNN-based ST-IDW Interpolation**

  The deployment of massive numbers of sensors, many of them moving, can lead to skewed distributions of sensor observations over space and time. To deal with this type of skew adaptively, we introduce the dynamic ST a$k$-Shell operator. This operator locates a bounded number ($k$) of sample points within a bounded search radius ($r$) during an iteratively expanding ST search using the ST grid index and Nested Shell Template (NST) that terminates when $k$ is reached. The ST a$k$-Shell operator is able to adapt to data density, with $k$ serving as the termination condition when data are dense and $r$ when data are sparse.

- **Stream Query Operator Framework for Evaluating Value Predicates over ST Phenomena**

  We introduce the ST Predicate Stream Query Operator Framework (STP-SQO framework) with different algorithms for evaluating value predicate queries over the ST phenomena accurately in near real-time. This novel approach evaluates the predicates over the phenomena representation instead on directly filtering the raw sensor data stream to achieve higher accuracy of predicate query result. Overall, we introduce three different approaches described in the next three contributions:

- **Seed Selection and Region Growing ST-IDW Algorithm**

  We introduce a **seed selection** algorithm that identifies a set of 'seed (raster) cells' that have one or more tuples that satisfy the predicate. Using seed cells, two region growing algorithms are proposed: a) Breadth-First (BF) and b) Scan Line (SL) using ST-IDW. BF region growing expands each seed to a region using an breadth-first pattern, while the SL approach expands regions row-by-row. Both region growing algorithms reduce the number of cells that need to be interpolated in regions which do not satisfy the query and thus improve query evaluation time.

- **Tile-based Seed Expansion Algorithm**

  A **tile-based** approach is introduced for high resolution grids to implement a greedy algorithm that partitions the prediction grid into larger tiles; if a tile is queued for interpolation, all cells in that tile are interpolated. If a tile contains a seed, it is interpolated, including all eight neighboring tiles and their cells. The tile approach trades interpolating more cells for not having to keep track of a parallel expanding search as done in the BF and SL region growing approaches.

- **Informed, Phenomenon-Aware Seed Expansion Algorithm**

  The **phenomenon-aware** algorithm, which **uses the interpolated query result from the previous window for optimization** is introduced as an improvement to

the **tile-based** approach to prune the number of cells interpolated by using a different algorithm based on how a snapshot from the previous window is categorized. The raster query result is partitioned into tiles and each tile is categorized as being `Interior` (all cells satisfy query), `Boundary` (mix of cells that do and do not satisfy query), and `Exterior` tiles (no cells satisfy the query) to determine whether or not each tile needs to be expanded or interpolated based on its classification and the seed cells from the current window.

### 1.4.2 Summary of Results

#### 1.4.2.1 ST Interpolation Accuracy and Runtime

The STI-SQO framework was prototyped using Java and it can be integrated into an open source DSE. The prototype was tested using simulated sensor data streams of moving sensors sampling the air after the Fukushima nuclear event in March 2011. First, the algorithms were assessed under different parameter configurations, including radius $r$, $k$ nearness, ST-IDW power $p$, ST anisotropy ratio $a$, IC, timestamp of the phenomenon, rate of change of the phenomenon, number of sensors per window, and window size, to determine the configurations that yielded the lowest RMSE; these configuration were evaluated further for runtime performance. The parameters $a = 4$ and $p = 4$ were identified as the parameters for ST anisotropy ratio and ST-IDW power that yielded the lowest RMSE. Using an IC at the center of the window resulted in the lowest NRMSE. For a dataset representing a sparse street network, using $r = 48$ resulted in the lowest NRMSE and for the more uniform distribution of sensors in the Cambridge and Random data sets, $r = 16$ was the appropriate parameter for the most accurate representation. Using a radius larger than $r = 16$ with ST a$k$-Shell for the Cambridge and Random data sets does not negatively impact NRMSE as search is adaptive to the data distribution and terminates after $k$ tuples are found. Using $k = 16$ achieved the lowest NRMSE for all data sets with an IC at the end of the window.

Using $a = 4$, $p = 4$, $r = 48$, $k = 16$, an IC at the center of the window, and the ST a$k$-Shell algorithm for ST-IDW our performance results show that the framework was able to generate $500 \times 500$ pixel spatial snapshots of ST phenomena with an NRMSE under 0.19 in 2.5 seconds with over 250,000 tuples per query window using sensors located in a dense, urban street network (Cambridge data set) and freely moving in space (Random data set). For the Japan data set, using $r = 16$ took around 10 seconds and using $r = 48$ took around 55 seconds with an NRMSE of 0.3. This increase in runtime for the Japan data set results from the fact that a large portion of the observation area consisting of water where ST a$k$-Shell must expand to the full search radius. In an adapted algorithm, these areas can be omitted, and the remaining ares can be interpolated in near real-time.

#### 1.4.2.2 ST Predicate Evaluation Accuracy and Runtime

Secondly, the STP-SQO framework was tested with regard to evaluating predicate queries. The algorithms for detecting and interpolating predicate regions were first evaluated with respect to the accuracy of the region they extracted compared to the region extracted by applying the predicate to the ground truth. The differences between these results were calculated using the Jaccard distance [74] and showed that the proposed predicate evaluation algorithms did not sacrifice accuracy compared to the baseline Naïve approach. Further performance testing was conducted to evaluate the runtime performance of the proposed predicate evaluation algorithms, which showed that the Scan Line region growing algorithm was the fastest predicate evaluation algorithm tested for all data set sizes and characteristics. Using the Scan Line algorithm for the ST Predicate Region Evaluator Operator and with ST interpolation performed by the ST a$k$-Shell algorithm, predicate evaluation takes around 0.2 seconds to evaluate a predicate for both an urban and rural street networks, for over 250,000 sensor observations per window from mobile, asynchronous sensors for a predicate result that takes up 28% of a raster with a resolution

over $500 \times 500$ cells. The Scan Line algorithm, which grows regions from seed cells line by line, was the fastest predicate evaluation algorithm for all tested configurations.

## 1.5 Outline of the Thesis

The remainder of the thesis is organized as follows:

Chapter 2 reviews the related work in the domain of near real-time processing of sensor observations of continuous ST phenomena and critically analyses the limitations of other approaches for capturing ST continuity of phenomena.

Chapter 3 introduces a system architecture and index structure for real-time indexing of streaming ST sensor observations.

Chapter 4 introduces an approach for generating spatial snapshots of a phenomenon computed using ST interpolation and a sliding data window over a stream of sensor observations of the phenomenon.

Chapter 5 introduces approaches for evaluating predicates over ST phenomena.

Chapter 6 contains conclusions and directions for future research.

## 1.6 Intended Audience

This dissertation is intended primarily for researchers and developers interested in evaluating queries over continuous, ST phenomena in near real-time using a DSE, especially those interested in implementing support for ST fields, ST interpolation, and ST predicate evaluation. The intended audience includes experts in the field of computer science whose research focuses on real-time processing of spatial and ST data. The thesis is also relevant to specialists of GIS and scientists working on modeling environmental phenomena and looking to incorporate real-time sensor data into models.

# CHAPTER 2
# BACKGROUND

This chapter describes the background and reviews the related work with regard to querying continuous Spatio-Temporal (ST) phenomena observed via point-based sensor data streams originating from large deployments of wireless, network-connected sensors in real-time.

In this chapter, we first define the terminology around ST continuous phenomena. Secondly, we review the limitations of processing such phenomena using methods and techniques found in traditional information systems, namely Geographic Information Systems (GISs) and Relational Database Management Systems (RDBMSs). Thirdly, we introduce Data Stream Engines (DSEs) and their novel architectural characteristics that support real-time query processing for massive data streams. We conclude this chapter with an assessment of the current limitations of support for ST stream query processing in DSEs.

## 2.1 Spatio-Temporal Continuous Phenomena in Spatial Information Science

In this section, we discuss the distinction between different entity types in spatial information science, and focus on physical entities that are *continuous* in space and time, so-called fields. After formalizing the description of such entities, we analyze how they are represented and processed with state of the art GISs and RDBMSs.

### 2.1.1 Terminology

#### 2.1.1.1 Fields vs. Objects

Entities in the domain of spatial information science are classified into two categories: *fields* and *objects*. In many cases, geographic entities can easily be identified as one type or the other, while with some entities it depends on the perspective whether something should

be classified as an object or a field. Defining the difference between objects and fields, Galton [52] states that "objects often involve a higher level of abstraction than fields: fields provide the raw data in the form of what properties are ascribed to individual locations, whereas objects are 'chunks' that are 'carved out' of the raw data and referred to as unities with their own (higher-level) properties."

Considering space as being a bounded geographic area, we use the example of a spatial field and spatial object. We define a field as a property value (or set of values) of specific locations within the space. For example, considering a geographic area, temperature can be measured at any location within a well-defined spatial region. Thus, the temperature field is continuous over space and bound by the spatial region. On the other hand, objects are entities with distinct identities and have property values that depend on the object; for instance, a lake is a spatial object. In some cases, there is a choice whether an entity should be represented as either a field or an object, for instance, a hurricane. As a field, the spatio-temporal behavior of a hurricane is captured by one or more location dependent properties (e.g., barometric pressure, wind speed, cloud cover) over a region. On the other hand, as an object a hurricane is represented by a well-defined boundary and identity-based properties (e.g., maximum sustained wind speed, eye pressure, speed and direction of movement.) Galton [52] identifies these types of entities, which can be viewed from the perspective of both a field and object, as hyperobjects or multi-aspect phenomena.

Objects and fields are referred to as *continuants* and *occurrents* in philosophy [52], respectively, by how they exist in time [61]. Examples of continuants are people, carts, trees, buildings, roads, mountains, and administrative units [52]. Examples of occurrents "include: your smiling, her walking, the landing of an aircraft, the passage of a rainstorm over a forest, the rotting of fallen leaves" [61], which are examples of processes. Continuants persist for a period of time while occurrents evolve as a process.

In this dissertation, the focus is on **fields**, specifically fields that are continuous over time and space (e.g., environmental phenomena such as the temperature value over the Earth's surface.)

### 2.1.2 Formal Definition of Fields

As discussed above philosophically, fields represent phenomena that changes over time. To capture a field's properties formally, we introduce a mathematical equivalent [51, 52, 38, 91]. Using a mathematical foundation allows us to describe the characteristics more systematically, and it provides a well-understood, formal foundation for implementing fields in information systems [90, 91, 26].

#### 2.1.2.1 Mathematical Definition of a Field

Galton formally defines a field over a space $S$ and states that "a field over $S$ is a (possibly partial) function $f : S \rightarrow V$ , where $V = V(f)$ is the set of *values* characteristic of the field. The value set may be ordered or unordered, finite or infinite, continuous or discrete, numeric or symbolic" [51]. Galton makes no restrictions on $S$, just stating that it requires "some sort of topological or pre-topological structure ascribed to it" able to support connected components [51]. This means that $S$ "may be discrete or continuous, finite or infinite, and it may or may not have various homogeneity or isotropy properties" [51]. Without restricting the space, $S$, a field may be used for different spaces, applicable examples being geographic space, time, the combination of geographic space and time through restrictions on the space.

#### 2.1.2.2 Spatial Field

A **spatial field**, short for geospatial field, $f_s$ is a mapping from geospatial locations to values and represents the value of a variable over space. In the scope of this thesis, a spatial field will be assumed to be two-dimensional and the spatial domain, $S$, is assumed to be planar and defined formally as $S \subseteq \mathbb{R}^2$. The spatial field is thus defined as $f_s : S \rightarrow V$.

#### 2.1.2.3 Temporal Field

A **temporal field** $f_t$ is a mapping from time locations to attribute values that captures the change of an attribute over time. For a temporal field, the temporal domain, $T$, is assumed to be one dimensional, linear and formally defined as $T \subseteq \mathbb{R}$. Similarly to the definition of spatial fields, a temporal field is defined as $f_t : T \rightarrow V$. A temporal field can represent the change of a sensor's location or any other type of measured value over time.

#### 2.1.2.4 Spatio-Temporal Field

A Spatio-Temporal Field (ST field) has a domain $D : S \times T$, with $S$ and $T$ as defined in Section 2.1.2.2 and Section 2.1.2.3, that is the domain contains space-time locations. A ST field is defined as a function $f_{st} : S \times T \rightarrow (V \cup \emptyset)$ that assigns each element of the cross product $S \times T$ an attribute value in $V$ or a null value $\emptyset$. The null value is needed for cases where the ST field is not continuous.

The phenomena of temperature over a city and how it changes over time is an example of an ST field that is continuous over both space and time. Here, every $< s, t >$ location pair maps to a temperature value. Sensors around the city sample the temperature at discrete points in space and time, which can be conceptualized as a discrete ST field. Similarly, an ST field can be used for discrete, ST point events that occur irregularly in space and time (e.g., location and time of lightning strikes). For an irregular, discrete phenomena, taking the cross product $S \times T$ yields space-time locations, $(s, t)$, where an event (e.g., lightning strikes) did occur. To handle this issue, the option of mapping to either $V$ or `NULL` was added by Liang [90].

#### 2.1.2.5 Specialization of Spatio-Temporal Fields

In the scope of this thesis, we will discuss three different types of ST fields: 1) fields as human and formal descriptions of continuous phenomena, 2) fields observed at ST point locations by sensors, so-called observation fields, and 3) a 'continuous' computer representation of the phenomenon, the so-called continuous spatio-temporal field.

- **Phenomenon**

  The phenomenon being observed is the field as it exists in the physical world, termed physical field in [78]. In this thesis the assumptions about the physical field that is the phenomenon are that it has:

  - continuous spatial domain
  - continuous temporal domain
  - continuous value range
  - bound to a spatial extent of the earth
  - not bound to a temporal interval

- **Observation Field**

  A phenomenon can be captured using sensors; however, sensors can only sample at discrete points in space and time. The *observation field* is the discrete field that contains the observations of the phenomenon over a period of time and can be thought of as a point cloud, a Spatio-Temporal "data cube" in which two of the cube dimensions are spatial and the third is temporal [136]. The restrictions for the *observation field* are:

  - contains a finite number of observations
  - discrete spatial domain
  - discrete temporal domain
  - continuous value range (though limited by numeric precision)
  - bound to a spatial extent that is a subset of the surface of the earth
  - bound to a temporal interval

- **Continuous Spatio-Temporal Field**

  The *continuous spatio-temporal field* provides a continuous view of the *phenomenon* by combining the *observation field* with an estimator function. This term is in regard to a computer representation and approximation of the real-world phenomenon. The estimator function calculates a value for any ST point using data from the *observation field*. The *continuous spatio-temporal field* can then be queried to get an estimated value for any point. The restrictions for the *continuous spatio-temporal field* are:

  - can process finite number of prediction points in finite time
  - continuous over the spatial domain (though limited by numeric precision)
  - continuous over the temporal domain (though limited by numeric precision)
  - continuous over the value range (though limited by numeric precision)
  - bound to a spatial extent is a subset of the surface of the earth
  - bound to a temporal interval

A major challenge of creating computer representations of continuous fields is dealing with the continuous nature of the phenomenon in the physical space and finding precise representation or approximations for them on a computer. Therefore, we will discuss current approaches of such representations first.

### 2.1.3 Representations of Continuous Space and Time

The mathematical definition of fields provides a precise foundation to formalize fields and capture the aspects of spatio-temporal continuity mathematically. In detail, this means that ST *phenomena* are defined at infinitely many spatial locations and time instants, even with bounded space and time. A characteristic of such continuity is that we can 'zoom in' infinitely and 'see' more detail of the phenomenon. Today, two different types of computer representations for fields exist: *equation fields* and *sampled fields* [91]. An equation field is the implementation of the mathematical, predictive function that encodes the

well-understood, entire behavior of the phenomenon. For example, gravity can be seen as an equation field. For equation fields, observations are not necessary. On the other hand, a sampled field relies on observations and a prediction function that estimates the value of the phenomena for non-sampled points based on the sample and some statistical prediction.

In the context of representing continuity of phenomena on a computer, equation fields can support the characteristic of 'infinitely zooming in' and providing more detail since they are defined via a computational function. The function delivers a value for any input value of the domain. Similarly, sample fields can represent continuity, and any requested value is estimated based on the samples. So, zooming might be possible but is constrained to the available sample to produce a reasonably accurate result. Visually representing the entire field, in both cases, needs to deal with the fact that only a finite number of points can be stored on a computer, and if a field is represented as a raster, a finite number of cells needs to be chosen.

#### 2.1.3.1 Defining Continuity

While continuity is easy to define mathematically, there are more considerations when defining continuity in the context of space and time and for ST phenomena. The underlying concepts of continuity are defined below.

- **Boundedness vs. Unboundedness**

  A finite number of sensors with finite sample rate can take a finite number of observations of a continuous ST phenomenon over a finite period of time, while the phenomenon itself is defined at infinitely many space-time points. Such a phenomenon is continuous over space, time, and value but it can be only observed and represented within a study area, that is, within a spatially-bounded study area $A \subset \mathbb{R}^2$ and time interval $t \in T \subset \mathbb{R} | t_{start} \leq t \leq t_{end}$ and it is defined at points within $A \times T$. If $A$ or $T$ contains infinitely many points, then there are infinitely points in $A \times T$.

- **Continuous vs. Discrete**

  Between two (non-equal) points in an ordered domain, the number of points will be finite or infinite depending on the domain of the points. If the set of integers ($\mathbb{Z}$) is used, then there is a limited number of points between the two points in question. If the set of all real numbers ($\mathbb{R}$) is used, then the number of points is infinite, as between every pair of points there exists another point.

  The set of real numbers is continuous, since between any two numbers another number always exists, whereas the set of all integers is discrete because between consecutive integers, (e.g. 1 and 2), no other integer exists. A continuous set is always infinite because of this fact, and a discrete set can be finite or infinite depending on whether or not it is bounded. In the case of a field, the domain can be continuous or discrete. An observation field has a discrete domain, while a continuous ST field has a continuous domain.

- **Resolution and Granularity**

  Resolution is either the measurement or representation of precision for space, time, and value. The resolution of each dimensions is limited by the measurement instruments used and the chosen computer representation. For data that are continuous (real numbers), floating point number representations are commonly used for their ability to represent (approximate) decimal numbers with different exponential scale. If there is a limit on how fine the resolution between elements can be, such as the spacing between consecutive integers, clock ticks, or pixels in an image, that resolution limit is called the *granularity*.

Mathematically, the set of real numbers is continuous, as was described above. Extending this concept of continuity from a numerical dimension to a plane or cube (as could be used to represent an space-time cube) is intuitive in theory. Space-time can be thought of as being continuous, except at extremely small distances where it is unknown

whether space-time is continuous or discrete [5]. Some phenomena that appear to be continuous, (e.g., temperature and humidity) have a resolution limit, as the phenomenon appears continuous at the scale of human perception, but it is undefined at the molecular scale in the vacuum between individual molecules. However, in the context of geographic information science, this fine of a scale is not necessary and for all purposes temperature and humidity can be considered as being a continuous phenomenon.

There are inherent challenges with representing continuity on computers. All numbers must be represented using bits, and computers can only store and process a finite number of bits. With a limited number of bits, only a subset of the real numbers can be represented. Even with the inherent resolution limits of number systems and their limited number of values, computer representations of decimal numbers in practice are sufficient and exceed the precision of measurement equipment and appear to work as if a continuous number system were used. The continuous ST field model can be thought of as being continuous over space and time, but does not work for exceedingly small distances because of possible granularity limits on physical space-time, the phenomena may be undefined between molecules, and limits of computer numeric representation.

The computer representations for spatially continuous phenomena are well established; however, temporal and ST continuity is less common in spatial information systems.

#### 2.1.3.2 Representations of Spatial Continuity

Today, a variety of computer representations for continuous spatial fields exists. Continuous phenomena are typically observed either remotely using imaging sensors or in situ using stationary or moving point sensors. Representations of continuous spatial phenomena can be classified in raster-, vector-, and field-based.

- **Raster**

  A raster [22, 35] representation uses a regular grid consisting of equal-sized polygons. Most commonly, the grid is composed of square *grid cells*, or *pixels*, but other

alternatives include triangular or hexagonal cells. A value is assigned to each grid cell that captures the value of the phenomenon within that cell. The value may be an observation (e.g., the nearest observation to the cell center) or a calculated aggregate based on a set of nearby observations.

- **Vector**

  Vector [22, 35] representations include contour lines and tessellations created using triangles or other polygons. The most common tessellations are Triangulated Irregular Network (TIN) [131] or Voronoi diagrams [18, 23]. A TIN is a triangular mesh, created by first identifying the significant points (including local maxima and minima and saddle points), and then using this subset of points non-overlapping triangles are constructed using three neighboring points [131]. A TIN is a continuous surface where within each triangular shaped facet the value is estimated by linear or cubic polynomial spatial interpolation using the values of the vertices of the triangle [84]. Each polygon in a Voronoi diagram contains one sample point and the value of the interior of the polygon is the value of that sample point. Thus a Voronoi diagram has a step change between adjacent polygons having a different value while a TIN has a continuous change along each face.

- **Continuous Spatial Field**

  A data model based on the concept of fields [26, 78, 90, 91] provides a continuous view of space and time. Kemp [78] argues that "the simple concepts of raster and vector may be incomplete for working with representations of continuous phenomena. There is more to representing reality than just breaking it into pieces that can be fit into the computer." Camara et al. [26] calls for a field data model as data structure, independent from any type of vector or raster representation. Liang et al. [90, 91] extend this perspective and define a field data model to support streaming data, fields that change over time, and operations over fields. The basic idea of a field Abstract

Data Type (ADT) is that various representations (such as rasters, contour lines, or TIN) are computed on the fly, and the field is not locked into a single representation.

For all three representations, a finite set of user-provided sample points are used to construct a continuous representation. While with a TIN and Voronoi diagram the process of transforming sample points into a continuous representation is tied to the representation format, with a raster there are many different algorithms from which to choose. Similarly in Camara et al. [26], the user can specify the estimator function used to calculate a value for a desired query point. The transformation of a set of sample points into a continuous representation is performed by a spatial interpolation algorithm [85, 84], to be discussed later in Section 2.1.4.2.

#### 2.1.3.3 Representations of Temporal Continuity

The time domain has gained increasing importance in GIS, with the availability of inexpensive sensors with wireless communication that are able to observe continuous, high resolution temporal change. Initial work in Spatio-Temporal entities focused on capturing and preserving discrete changes of spatial objects, such as the ownership change of a land parcel or the development of land over time. However, slow, gradual change of environmental phenomena such as geological phenomena (e.g., orogenesis and erosion [36]) or climate change [130] have also been of interest. These are gradual changes that take place over long time periods, such as years, decades, or centuries. Today, with the availability of live streaming sensors a new type of temporal continuity and rate of change can be observed, including the individual trajectories of cars and the continuous change in a phenomena over time at the location of sensors. Models of time have several characteristics including structure, continuity, density, and granularity [129, 153].

**Structure:** In the physical world, humans perceive the *structure* of time to be linear and progressing onward like an arrow into the future. In a computer, the representation of time structure is more flexible. For example, if applications are used to predict future

states, a *branching time model* captures multiple future possibilities from the current time. A periodic (also called cyclic) time model can be used to represent cyclical processes (e.g. lunar cycle, tide, a week).

**Continuity and granularity:** Time is perceived as being *continuous* in the physical world: between any two time instants another time instant exists [153]. Computer clocks and representations of time provide a high resolution, but are discrete. Clock instruments produce discrete pulses at a constant frequency, and computers are only able to tell that an event occurred between clock pulses rather than the exact time. The term *chronon* is used to describe the smallest, indivisible unit supported in a system [75]. Computer data types for representing time have a *granularity* for temporal resolution. For instance, a `date` data type has a granularity of a day while a `timestamp` can have a *granularity* of a second. A point in time is represented in a computer using a finite number of bits, where the required size in bytes to store a time point is based on how the data are encoded in the representation, the range of the representable time value, and the *granularity*. Even though resolution limits to time exist for computer representations, time can be modeled as a discrete or continuous variable. Continuous time is best for fields, as it can capture the continuous change of a phenomena over an interval. Discrete time can be used for abrupt change, like the movement of an administrative boundary [171].

The terminology of *transaction time* and *valid time* is introduced by Snodgrass and Ahn [152] to distinguish the time point or time interval when information about an entity is present in a database system and the time point or time interval when the entity is valid in the real world. For sensing applications, valid time is the time stamp attached by a sensor, and database time the time stamp when information arrives at the information system. Therefore, a database system can reason about network latency and correlate different data correctly. Time in databases is discussed further by Snodgrass [153] and the concepts of transaction and valid time are extended by Worboys [171] to support recording changes to spatial objects over time in a database.

#### 2.1.3.4 Representations of Spatio-Temporal Continuity

In the past, the primary focus of Spatio-Temporal GIS and Spatio-Temporal Database Management System (ST-DBMS) has been mostly on spatial objects, and how to capture how they change and how their relationships with other spatial objects change, over long periods of time with few updates, and potentially establish a topological temporal framework for representing entity-based events [17, 36, 83, 170].

With the availability of inexpensive GPS units, database research started to support queries over the *continuous* movement of spatial objects over time [40, 47, 62, 168]. However, DBMSs could still only support sparse updates (e.g., once every 30 min). With the support of DSEs [155], the continuous movement of spatial objects over time can be supported in real-time today [48, 109, 119]. Models that attempt to capture continuity for fields over space and time can be classified as having a primarily spatial focus, temporal focus, or treating space and time equally.

- **Spatial Focus**

  Today, the most common view of Spatio-Temporal continuity is still the primarily spatial focus that captures the change of ST phenomena as a temporal sequence of *spatial* snapshots, called the the snapshot model [17]. The snapshot model represents space as being continuous, and time as discrete [54] and regularly or irregularly spaced time instants. A well-known representation of ST continuity using raster snapshots was provided by the space-time cube [54, 99]. It is common that the distance between snapshots is relatively wide compared to the change of the phenomenon. Therefore, the temporal aspect has a more 'discrete' feel than the spatial aspect, while both aspects are discrete in nature.

- **Temporal Focus**

  Other representations focus on ST phenomena from a temporal perspective, looking at the process of the phenomena over time and the times when change occurs. This allows a user to compress a temporally dense sequence of raster snapshots so that

change between rasters is captured but redundant information is omitted. Peuquet and Duan [132] looked at extracting interesting event regions from ST continuous phenomena (represented as rasters) in the Event oriented Spatio-Temporal Data Model (ESTDM). Since only incremental changes are stored, less data are kept overall. Each event is time-stamped and associated with a list of event components, which are the raster cells that changed from the last event.

- **Spatio-Temporal Focus**

  The continuous ST field [26, 90] provides a continuous view over both space and time. This allows a phenomena to be sliced as a temporal field of spatial fields or spatial field of temporal fields. Camara et al. [26] implemented spatio-temporal fields on top of the SciDB array database system using rasters and support operators for subsetting rasters, while [90] provides a prototypical mapping to data streams.

Mennis et al. [99] note that using a fixed ST resolution is disadvantageous for analysis across different datsets as the representation is inflexible and requires re-sampling as all data need to have the same spatial and temporal gird size to be comparable. This is the advantage of the continuous spatio-temporal field as the representation is not tied to a specific resolution.

So far, we introduced different concepts related to how scientists classify geographic phenomena into objects and fields, and how fields can be described more formally. We also discussed the limitations of representing aspects such as spatial and/or temporal continuity on a computer. In the following section, we discuss the current state of the art in sensing geographically continuous phenomena via point-based sensors and the methods of reconstructing a continuous computer representation.

### 2.1.4 Sensing and Reconstructing Spatio-Temporal Phenomena

Continuous phenomenon are often observed via camera-like instruments from space (remote sensing), but the focus on this dissertation is the sensing based on many individual

point-based sensors in physical space. Thereby, the layout of a sensor network can be spatially regular or irregular. Further, we discuss the different types of spatial interpolation methods [103] that are used to approximate and fill in missing values for a continuous representation, such as a raster.

#### 2.1.4.1 Sensing Spatio-Temporal Phenomena

Continuous ST phenomena can be sensed using in situ sensors that sample periodically. Depending on the type of sensor and an application's needs, different spatial representations are more suitable. A raster data format is commonly used in remote sensing, especially with cameras and line scanners. However, to simplify data integration, rasters are also common formats for in situ based sensing. To represent data from in situ sensors as a raster, spatial interpolation is used to fill in unsampled cells in a raster using the available sensor observations.

#### 2.1.4.2 Reconstructing Continuous Phenomena using Spatial Interpolation

When irregular spatial sampling is used, spatial interpolation is needed to transform the set of point observations, *observation field*, into a continuous spatial representation. Spatial interpolation methods can be classified as being non-geostatistical, geostatistical, or as more advanced combined methods that use both non-geostatistical and geostatistical methods [84, 85].

Non-geostatistical interpolation methods such as IDW [149], TIN [131], or Voronoi [18, 23] do not consider the statistical properties of the distribution of sample points and are not able to assess how well the interpolation fits the data.

As an example, IDW is based on the premise that the value at a non-sampled point can be approximated as a weighted average of values at points that are spatially close, due to Tobler's Law [159]. To assess the *prediction error*, the error between a value predicted using interpolation and the actual value of the phenomena at that point, external analysis is required. For example, the error in a non-statistical method can be calculate by omitting

some sample points and checking the variation of the interpolation at those points [101, 112, 160]. In the ideal case where the ground truth of a phenomenon is known, possible when using simulated sensors that sample a simulated phenomenon on a computer, the difference between the interpolated and known value can be determined for each raster cell and then aggregated across a raster to calculate the RMSE.

Geostatistcal methods (e.g. Kriging [80]) assess the spatial (or the temporal) correlation of observations and calculate the uncertainty of each predicted value as the *kriging variance*, thus providing a continuous map of both the interpolated value and uncertainty [96]. In contrast to IDW, spatial Kriging assesses the spatial autocorrelation of sample points, using a variogram. Ordinary Kriging is referred to as the best linear unbiased estimator [154]. However, this comes at the expense of increased computational complexity for Kriging ($O(n^3)$) [96, 176] over IDW ($O(n)$).

#### 2.1.4.3 Reconstructing Continuous Phenomena using Spatio-Temporal Interpolation

The problem of ST interpolation is relatively new, and only limited related work exists for estimating phenomena that are continuous in both space and time based on point observations [46, 87, 88, 154]

In [87, 88], two alternative approaches for ST interpolation are considered. In the 'reduction' approach [87, 88], for each sample point location a function is generated that captures how the value of the phenomenon changes over time using the time series of sample points and 1-dimensional interpolation to predict a value at the sample point location for any desired time instant. These functions over time are queried at a time instant and processed using spatial interpolation to calculate a value for any ST point. In the 'extension' approach [87, 88], 2-dimensional interpolation methods are extended to 3-dimensional space where time is treated as the third dimension. Treating time as a third dimension has yielded the best accuracy [87] and been the approach used by

others [46, 154]. An issue with treating time as a third spatial dimension is that the autocorrelation of a phenomenon between the spatial and temporal dimensions, and an anisotropy factor can be used to scale distances in spatial and temporal dimensions [82] .

Different ST interpolation algorithms include shape functions [87, 88], Kriging [46, 87, 154], and IDW [87, 46]. Shape functions [87] come from finite element methods, and produce a vector mesh that represents the value of a phenomenon. In contrast, Kriging and IDW are typically used to interpolate regular grids. Time is added as an additional dimension in ST Kriging [46, 87, 154]. Both [46, 87] investigate ST IDW; however, [87] does not consider anisotropy between space and time.

### 2.1.5 Critical Review

Over the last decade, sensing continuous phenomena has significantly changed due to increasingly dense sensor deployments and an ability to observe phenomena and transmit the observations in real-time.

Traditional GIS and RDBMS technology have struggled to keep up with the increasing throughput demands from these new sensor environments. To work with sets of point sensor observations, spatial interpolation is needed to generate a continuous spatial representation and ST interpolation is needed for a continuous ST representation. Spatial interpolation is well established, but ST interpolation isn't as well established. Faster variants of interpolation algorithms are necessary to work with live sensor data. Existing interpolation methods focus on producing the best approximation of a phenomena using a limited data set where processing time is not an issue. This is different from the situation where there are live sensor observations and algorithms must keep up with the massive amount of continually arriving sensor data.

## 2.2 Data Stream Processing Support for Spatio-Temporal Phenomena

As required by the real-time processing and throughput demands of massive sensor deployments, financial transactions, and fraud detection, a significant part of database research has focused on developing a novel type of data management technology called DSEs over the last 15 years. DSEs have been designed to cope with extremely high throughput of tuple updates, bursty data arrival, and real-time answers of queries that are executed continuously over new arriving data. In DSEs the traditional RDBMS bottleneck of writing data to disk before it can be processed in queries is eliminated. In a DSE, data are both stored and processed in main memory [156].

### 2.2.1 Data Streams

Similar to RDBMS, DSEs are structured around a relational or tuple-oriented data model. A *tuple* is an element of a data stream and represents an update about the phenomena represented by a stream. A tuple can represent the price update for a stock ticker symbol, the time and corresponding location for a specific moving object, or the time, location and value of a sensor observation of an environmental phenomena. A tuple may have several attributes, possibly including a key identifying the item (e.g. an identifier for an stock, moving object, or sensor) or unique identifier.

A data stream is a sequence of tuples that are transferred incrementally to a receiving computer system, a *consumer*, for processing. The order of tuples in the stream is determined by their arrival time. Data streams are *produced* by a variety of external sources, especially sensors. A DSE is a *consumer* of the stream and decoupled from the *producer* and has no control over the number of devices streaming tuples, the arrival rate of tuples, or the arrival order of tuples. Once an element from a data stream has been processed by a DSE, it is discarded or archived.

In a data stream tuples arrive continuously, that is the next tuple may arrive at any time in the future. The arrival rate of the number of data tuples of any time point in the

future is uncertain. Therefore, a stream is called unbounded, that is, it is impossible to place a *bound* on the number of tuples that will have arrived by any point in the future [14]. However, in hindsight, the number of tuples that have arrived is finite.

In a RDBMS, a query is posed over a stored relation, that is, all tuples that are relevant for the query are available at the beginning of query, and they have to be pulled from the disk. In a DSE, a query is issued over a data stream, that is all data are not available yet when the query is issued, but continue to arrive during a query. Furthermore, a query is typically not posed over an entire stream but over a specified moving interval of a stream.

### 2.2.2 Stream Windows

To pose queries on a bounded set of tuples, the concept of 'windows' has been introduced with regard to stream query execution [14]. A window is often specified using either a fixed number of tuples (count-based window) or a time interval (time-based windows). If no window is specified, then the query is executed on each tuple individually. A window query starts executing with the opening of a window. Non-blocking query operators execute a tuple at a time (e.g a filter that selects tuples satisfying a predicate), while blocking query operators (e.g. a join between two streams) may begin when the first tuple arrives but wait to process until all tuples in a window have arrived to produce a result. The variables inside an operator are called the *operator state*.

The most common window types are windows that are defined by their *length*, i.e. count-based and time-based windows; other window types like the predicate-based window exist [58]. A count-based window defines the window length via a user-defined number of tuples. A time-based window, on the other hand, is defined by an time-based interval, and the number of tuples that arrive within the time interval can vary. Another important window parameter is the *slide* of a window. The window can either move continuously over the time (or tuple arrival) axis, as in [14], or periodically according to the window slide parameter.

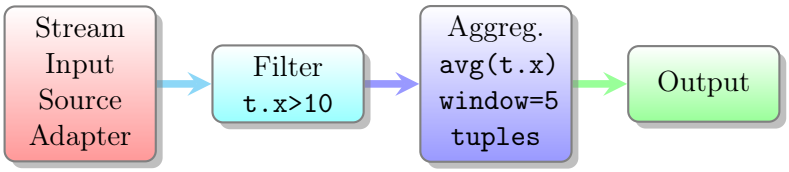



Figure 2.1. Stream operator graph

Based on both the length and the slide, windows are classified as *sliding*, *tumbling*, or *jumping* [59]. A sliding window is defined such that the slide is less than the length, a tumbling window is defined when the slide and length are equal sized, and a jumping window is when the slide is greater than the length.

### 2.2.3 Stream Query Processing

The query language for a DSE is either implemented as an extension of SQL, e.g., CQL [14, 13], or as an operator graph as in [30, 29]. The execution of stream queries is significantly different from processing queries over relations [19]: Stream queries are issued as continuous queries [158], that is the user submits the query once, but the DSE continues to re-evaluate the query over windows until it is explicitly removed from the system. Depending on the type of stream operators executed in a stream query, the query delivers a (new) stream of (filtered) tuples or an aggregated result tuple; this stream can be the input to other queries.

Similar to RDBMS, a query in a DSE is transformed into a query plan which is a graph of query operators that evaluate the query over one or several data streams. An operator may perform a select, project, join, aggregate, or other operation. Besides the operators, the query plan contains the connections between the operators representing the data flow between operators. The operator and the flow of data between them follows a Directed, Acyclic Graph (DAG). An example operator graph for a stream query is shown in Figure 2.1 for a query to pass tuples every tuple `t` where attribute `x` is greater than 10 and then aggregate as an average of `x` using a window of 5 tuples (slide of 1).

DSEs have several other components of built-in functionality to adapt to high data velocity that exceeds the processing capability. These components support adaptive resource handling to adapt to sudden and prolonged data bursts using load shedding [157], query plan migration and adaptive processing [140], and others.

The research field of data stream processing has advanced to the point that research prototypes and concepts have been incorporated into commercial products and there are multiple open source and widely used stream processing products. Notable DSEs available today are: Microsoft StreamInsight [6, 77]; Oracle Streams [53]; IBM System S [55] and InfoSphere Streams; Apache Storm [12]; Apache Spark Streaming [11, 174]; Apache Samza [10, 124]; and Apache Flink [9, 28].

### 2.2.4 Supporting Spatio-Temporal Streams

In a ST data stream a tuple has both a spatial location and timestamp attribute, sometimes called a geo-stream [71, 122]. Two types of ST streams are location updates for moving objects and updates of the value of point sensors, where the sensors may be stationary or mobile. Early research in ST DSEs [65, 105, 106, 109, 119] focused on supporting queries over moving spatial objects. In recent years, processing support for producing a representation of a continuous phenomenon from a stream of sensor observations data streams in near real-time has become available [7, 95, 96, 123, 176, 177].

#### 2.2.4.1 Stream Support for Moving Objects

Real-time moving object tracking has been a widely researched application of DSEs. Research has focused on tracking moving objects and their trajectories and analysis of traffic jams and traffic behavior in real-time [105, 106, 109, 119, 65]. Research systems are PLACE [109, 105], OCEANUS [50], and CAPE [119]. Commercial systems extension include GeoInsight which extends Microsoft's StreamInsight [6] to support online analysis for moving objects [77, 76].

While processing moving object stream queries is different from queries over fields, there are interesting lessons to be learned in how ST stream query processing differs from conventional stream query processing. The first significant difference is that all incoming update streams of moving objects are shared between all user stream queries. Secondly, queries are spatial in nature, for instance, monitor for each moving object the k-Nearest Neighbors (kNN) neighboring moving objects. Since a tuple could end up being compared with every other tuple in such a query, a naïve implementation that compares every element with every other element would have computation complexity $O(n^2)$. Therefore, to answer such queries fast, tuples are re-sorted in a spatial dimension instead of the temporal dimension (their arrival order), and main memory indexes are necessary [105, 106, 109]. Third, [77, 76] showed that real-time stream queries cannot achieve sufficient throughput if DSEs are used with traditional ST-DBMS libraries [77, 162]. Novel, single pass, stream based operator re-implementations are necessary to achieve the necessary throughput for real-time query answers.

Research has focused on extending stream data model and query languages with concepts for ST streams [48, 71, 81]. Galic et al. [48] presents a formal framework of data types for geo-spatial data streams and changing vector data types including trajectories of moving objects.

#### 2.2.4.2 Stream Support for Continuous Phenomena

Today, there is limited related work with regard to using DSEs to process sensor data streams to monitor continuous phenomenon. Liang et al. [90, 91] have focused on extending stream data models and query languages with a novel field data type hierarchy to support expressing high-level abstractions of ST phenomena on a schema level. Furthermore, [123] established the initial work exploring the feasibility of extending a DSE to monitor continuous phenomena in real-time based on massive sensor data streams. [123] explored novel stream based operators that implement the IDW method to generate spatial

raster snapshots of continuous phenomena. In order to make the problem tractable, simplified assumptions were made such that all sensors produce time synchronized updates. [123] demonstrated that a stream-based operator frameworks for spatial interpolation can achieve spatial interpolation of up to 256K update/s into a raster representation in under 2 seconds.

Aref [7] introduced a phenomenon-level monitoring framework, Nile-PDT, that is based on the assumption that groups of streams showing similar behavior over a period of time establish a 'phenomenon.' Phenomena are detected by joining observations from different streams using an SN-Join, an extension of the multi-way join algorithm MJoin [164]. This type of join, however, being agnostic of spatial neighborhoods, becomes too computationally expensive for large numbers of sensors. Additionally, each phenomena is represented as the set of sensors having similar values and is visualized as a closed polyline, which does not capture the spatial or temporal continuity of the continuous phenomenon being observed. The focus of our work is on representing the entire continuous phenomena, not just the boundary of phenomena.

Recently, [95, 96, 176] explored adapting Kriging [80] to work in a stream-based processing framework. Kriging has a computation complexity of $O(n^3)$ where $n$ is the number of observations [176] compared to other methods like IDW that have a linear time complexity $O(n)$. Zhong et al. [176] present a stream-based version of the Kriging spatial interpolation method for processing streaming data using sliding windows that is able to reduce computation when there are the same tuples in successive windows and where the set of sensors and their locations is relatively static. The resulting computational complexity after the optimizations approaches $O(n^2)$. Lorkowski and Brinkhoff [95, 96] present an approach that partitions sensor observations into subsets, creates submodels using the partitioned data, and then merges the submodels according to probability weight. Using the partitioning method, computational complexity is a floor function that increases roughly linearly with the number of observations. Both of these approaches have worked

with relatively small observation sets, with Lorkowski and Brinkhoff [96] processing 400 point observations in roughly 30 seconds and Zhong et al. [176] with a processing time of 20 seconds for the 600 initial observation points. Though these methods are more accurate than IDW used in [123], they have not been shown to be scalable to massive numbers of sensors of over 100K samples per second, that are spatially and temporally dense.

## 2.3 Summary

In summary, traditional GIS and RDBMS technology is unable to keep up with processing streaming data [155]. DSEs are a promising technology for representing and analyzing continuous phenomena observed by mobile point sensors that sample asynchronously, but within DSEs there is limited support for representing and analyzing continuous ST fields. Current DSE approaches [95, 96, 176] support monitoring of continuous phenomena and representing them as a raster, but these approaches, relying on Kriging [80], do not scale to be able to transform a massive volume of streaming of point observations from mobile sensors that sample asynchronously in near real-time into a spatially-continuous snapshot representation of the phenomenon that captures continuous change over time. Our previous work [123], supports spatial snapshots in near real-time using IDW for mobile sensors that update synchronously. New stream operator implementations are necessary to support representing and analyzing continuous ST fields efficiently in a DSE.

# CHAPTER 3
# A STREAM QUERY OPERATOR FRAMEWORK FOR CONTINUOUS REAL-TIME SPATIO-TEMPORAL INTERPOLATION

## 3.1 Motivation

Following the Fukushima Daiichi nuclear disaster in Japan in 2011, the Safecast [142] group of volunteers designed and built independent radiation sensor nodes to collect data about radiation levels in the area surrounding Fukushima rather than rely on the SPEEDI [34] data from the government of Japan. Equipped with a GPS receiver, the sensor nodes were used to collect measurements by volunteers driving in vehicles. Using mobile sensors, a much larger area can be sampled inexpensively compared to a stationary sensor network since measurements can be taken at different locations with the same sensor device.

One can imagine that in the future a much larger and denser radiation sensor network will be deployed, with the support of citizens and the government. Such a deployment can contain stationary sensors at street intersections and mobile sensors mounted on municipal and personal vehicles. The total sensor node population can easily reach around 1 million sensors for a metropolitan area. With such a large number of sensors it is unlikely that all sensors sample synchronously and instead massive streams of observations will arrive continuously at a centralized server where the data is to be integrated and analyzed, in near real-time.

At the central server, the constantly arriving individual observations need to be combined into a representation of the entire radiation field as it changes over space and time in real-time. Ideally, the observations streams are transformed into a 'movie' of how the radiation field changes. A 'movie' representation of the radiation field is a series of time-stamped spatial 'snapshots'. The spatial snapshots need to be recomputed often,

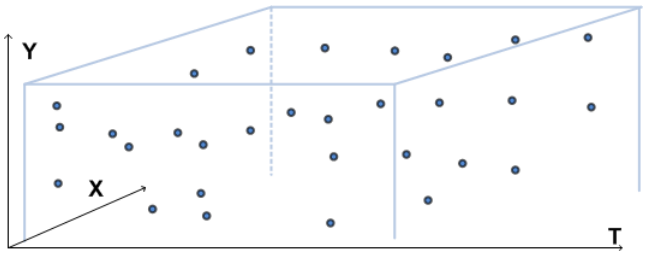



Figure 3.1. Sensor observation tuples distributed over space and time

potentially every 5 to 10 seconds, depending on the update rate of the sensor network. We call the rate at which new snapshots are generated the *prediction rate*. Each prediction interval consists of a finite set of spatio-temporal point samples, distributed over both space and time but bound by the spatial region of interest and prediction interval (see Figure 3.1).

## 3.2 Problem Statement

The example scenario, which is representative of many massive geo-sensor streaming applications, faces several challenges with regard to building a data management system to transform massive streams of continuously arriving sensor observations into a sequence of raster snapshots generated every few seconds to represent the ST evolution of a phenomenon for near real-time geo-visualization. We define the term 'near real-time' as being able to keep up with the generation of continuous representations based on arriving data with only 1-3 seconds of delay of the events happening in the physical world.

To constrain this research question, the following assumptions were made: sensors are *irregularly* distributed in physical space and send observations *asynchronously*. Sensors may be stationary, mobile, or transient, meaning that sensors may have a limited lifespan or enter or leave the spatial region of observation. Sensors are part of a massive sensor network (up to 1 million sensors total), the observations from each sensor are combined into one time-ordered stream, and the processing throughput target is 100,000 tuples/s.

The result of each prediction interval is at least one raster snapshot with a resolution of at least $500 \times 500$ cells.

### 3.2.1 Challenges

- **Asynchronous Sensing**

  With asynchronous sensing, the requirement that all senor nodes have a synchronized clock, which has shown to be a challenging problem in a sensor networks [43, 146], is removed. In a sensor deployment in which sensors likely have heterogeneous hardware, software, energy supply, and communications technologies and in which data processing is centralized and decoupled from the sensors, the centralized processing unit is unable to control when, where, or at what rate samples are taken, and thus, synchronous sensing is impractical. Asynchronous sensing leads to sensor observations being distributed over time and space, conceptualized as a ST point cloud as depicted in Figure 3.1, over a prediction interval. We assume that sensor network latency is minimal and that tuples are uniformly timestamped on arrival.

- **Mobile Sensing**

  Although many applications involve stationary sensors, in emerging scenarios (e.g., sensing urban air quality using sensors on transit vehicles) many of the sensors are moving [3, 173], and therefore, their location changes between updates. Statistical spatial interpolation methods determine the influence of each sample point on a predicated value based on its location. When sensors maintain the same location between consecutive predictions at different time stamps, the *influence factor* for each sensor does not need to be recalculated, and reduces a computationally expensive aspect of the algorithm. However, with moving sensors, the influence factors have to be recalculated for each individual interpolation step. This makes any spatial interpolation method that is computationally expensive with regard to calculating weights for predicted values not suitable.

- **Representing Spatio-Temporal Continuity of a Phenomenon**

  A prediction interval contains a set of discrete point observations over a spatial region and time interval, which establishes sampling a continuous phenomenon in this space-time continuum, while it changes. Similar to representing the spatial continuity of the phenomenon, it is challenging to represent its temporal continuity. While many approaches exist to deal with the spatial continuity, methods for representing temporal continuity accurately is a challenging, novel problem that today arises due to technological advances of wireless networked sensor networks producing high-density streams in real time. Ideally, the user would like to view a fine-grained temporal change of the phenomenon during a time interval. However, similar to spatial continuity, temporal continuity has to be approached via discretization. While ideally the ST continuity of a phenomenon is represented as a movie, even a movie consists of frames, i.e. snapshots, at discrete times with limited spatial resolution.

- **High Data Throughput Spatio-Temporal Interpolation**

  If sensor observations arrive at a very high rate and sliding query windows are short (e.g., 10 seconds or less) the interpolation step needs to execute efficiently to be able to keep up with the demands of continuously arriving data and transforming them into a continuous representation for each query window. The amount of execution time required to generate a raster at a given resolution using a spatial interpolation method depends on the algorithm implementing the statistical method. Typically, the run-time for an algorithm increases with the number of prediction locations and sample points. For instance, for $m$ prediction locations and $n$ sample points, the computational complexity for traditional IDW algorithms [149] is $O(mn)$ and Kriging [80] algorithms is $O(mn^4)$ and, even reformulated, still requires $O(mn^3)$ [68, 154, 176]. Therefore, traditional algorithms implementing spatial interpolation are not able to scale in order to deliver throughput for such high numbers of updates. Furthermore, the sample points within a window are distributed

over both space and time, and therefore require both novel Spatio-Temporal interpolation methods as well as algorithms supporting very high-throughput.

- **Spatio-Temporal Data Skew**

  An additional challenge is data skew, which is created during sampling and occurs due to the irregular geographic distribution of sensors, varying sampling rates, and potential movement of sensors. Thus, data skew can temporarily occur spatially, temporally, and spatio-temporally. *Spatial data skew* has the implication that the spatial distribution of sensors varies over the area of interest with respect to density per region. For instance, an area such as a dense urban center can contain a significantly higher number of sensors while in other regions few or no sensors exist (e.g., water bodies). *Temporal data skew* occurs when the number of observations per prediction interval changes over time. For example, sensors on public transportation buses sample only when the bus is operating. Also, since sensors sample asynchronously the density of observations within a prediction interval can be temporally skewed due to random effects. The combined ST data skew is described as a changing spatial skew over time or changing temporal skew over space (e.g., a swarm of aerial vehicles tracking a plume [145]). All three types of data skew can influence the data quality of the prediction raster generated using ST interpolation. Thus, an interpolation algorithm needs to be aware of and adapt to spatial, temporal, and ST data skew to improve the performance and interpolation quality.

- **Sliding Prediction Intervals over Spatio-Temporal Data Streams**

  Sliding intervals are defined as consecutive observation intervals that overlap in time. From a practical perspective, this means that a portion of the data from the 'older' interval are preserved for the current interval; sliding intervals allow the users to compute 'running' aggregates. To support such queries, it is necessary to be able to rapidly discern the data that are expired from an interval and the data to be carried

over to the next interval. This is a significant problem due to the very high data arrival rate in sensor stream environments.

## 3.3 Related Work

In this section, we review the related work with regard to processing massive ST point clouds of sensor observations and representing ST continuous phenomenon in real-time. Additionally, we explore how DSEs have been used for processing stream queries over moving objects streams and which lessons can be learned from this type of near real-time ST stream processing for ST continuous phenomena.

### 3.3.1 High-Throughput Spatial and Spatio-Temporal Interpolation of Massive Point Clouds

Although high-throughput Spatio-Temporal interpolation is necessary for transforming massive point cloud streams into continuous representations of phenomena, little progress has been made so far to address this. Therefore, we first investigate the current state of the art in purely spatial interpolation algorithms for high-throughput data processing, and then analyze the advancements in Spatio-Temporal interpolation.

#### 3.3.1.1 High-Throughput Spatial Interpolation

Traditional algorithms to implement spatial interpolation methods are designed to work with sparsely sampled data and in an offline context where the amount of runtime required is typically not critical. For novel applications with high data rates and fast throughput, diverse work has been proposed to adapt traditional algorithms for these requirements. We focus on interpolation methods that generate rasters as output and address the two main categories of such methods: adaptations for IDW algorithms and Kriging algorithms.

Several approaches addressed the problem of increasing throughput for IDW [123, 138, 68]. One way to improve the performance of IDW is to reduce the number of sample points that are used to interpolate a cell in the prediction grid. This can be

achieved by including only sample points within a specified radius around each prediction location [123, 138] or only using the $k$ nearest sample points to a prediction location [4, 123]. In [68], it is assumed that sensors sample synchronously and are stationary. Here, sensor locations and normalized observation weights for interpolation are stored in a matrix, and new observations are supplied as a vector; however, this approach is not feasible for mobile sensors as the weights have to be recomputed for each sensor that has moved. Another approach to speed up execution is parallel execution of IDW. This has been explored for CPU [68, 123], GPU [68], and clusters [70]. Our previous work [123] is the only approach that has investigated parallel, stream-based IDW for streaming point observations.

Other work has explored the adaptation of Kriging algorithms to increase throughput. Traditional implementations of Kriging [154] have a computational time complexity of $O(n^4)$; [68, 154, 176] reformulated the algorithm to achieve $O(n^3)$. However, scalability becomes the main challenge for real-time streaming applications as otherwise processing power $p$ must also scale by $p^3$ to keep execution time constant.

Zhong et al. [176] proposed an adapted version of spatial Kriging for stream processing that works for sliding, count-based window queries, which assumes stationary sensors. Here, the assumption is made that a portion of the data remains constant between consecutive windows and, thus, is reused. The amount of tuples added and removed (20 per window slide) is significantly smaller than the overall number of tuples per window (600 tuples). The incremental algorithm executes in roughly 10% of the execution time of the initial window for each window thereafter. For instance, with an initial execution time of slightly under 20 seconds, each of the subsequent windows takes around 2 seconds to compute for the following windows when using a $50 \times 50$ prediction grid. The resulting average computational complexity is $O(n^2)$. However, our work performs ST interpolation with a target throughput of 100,000 tuples/s and complete representations in about 2 seconds.

#### 3.3.1.2 High-Throughput ST Interpolation

The related work with regard to high-throughput spatial interpolation is very limited today due to the fact that both the statistical methods to address ST interpolation are in their infancy, and thus, algorithms to implement them are also very limited. Most of the work has been done in the area of extending spatial interpolation methods to incorporate time, e.g., ST IDW [87, 88, 46] and ST kriging [82, 103], however these do not address the throughput challenges for near real-time representation of massive sensor data streams.

For performing ST interpolation using ST kriging more efficiently, such as for ST sensor observation streams [95], modifications have been made to the algorithms [154]. The results produced by these algorithms are ST snapshots that are periodically completely [154] or incrementally in areas of change [95].

A challenge with ST interpolation of point observations is the anisotropy distance measured in space vs. time [82, 83]. Li and Revesz [87] use ST IDW and treat distance in space and time uniformly. Fitzner and Sester [46] discuss extending IDW for ST data points using a ST anisotropy factor and present a ST version of IDW able to use directional movement information from the phenomenon.

Lorkowski and Brinkhoff [95, 96] proposed a stream-based adaptation of ST Kriging to produce incremental updates to a raster snapshot for each new set of data points, assuming that sensors asynchronously sample at a slow rate (at most around 20 tuples per minute). Sensor observations are partitioned into subsets, and submodels are created using the partitioned data. Submodels are then merged according to probability weight. Using the data partitioning method, the computation complexity is a floor function that increases roughly linearly with the number of observations. The method was tested for a comparatively small number of observations (100-400 points). The 400 point set was partitioned into batches of 10 points; computational time took over 30 seconds for the combined result even using the proposed method for a $150 \times 150$ raster grid. Thus, this approach is not feasible for large numbers of mobile sensor stations.

### 3.3.2 Stream Query Frameworks for Moving Objects Update Streams

DSEs have been used to support real-time stream query evaluation over moving objects [107, 105, 106, 108, 104, 127]. Moving objects are represented as a point, and the spatial change of this point over time, i.e. a moving object's trajectory, is the focus. This problem is different from synthesizing point clouds into continuous representations at particular time stamps. However, some similarities between moving object stream query processing and continuous phenomena can be found. Typical stream queries for moving objects streams are "find all vehicles in a specified region;" to answer such spatial queries over sensor data streams, a shared, in-memory, spatial or ST index structure is proposed [105, 106, 108]. The need for a shared ST index across stream queries is similar to our work; however, for representing phenomenon in real-time, the purpose of the ST index is different since it needs to be designed to efficiently identify the sample points to use to interpolate each raster grid cell and is not concerned with tracking the movements of sensors.

Some SDBMSs include libraries supporting spatial queries. Since these spatial libraries have not been designed for the stream processing paradigm of incremental, single-pass processing, they cannot be used for efficient stream query evaluation [77, 162].

### 3.3.3 Spatio-Temporal Indexing for Stream Queries

Today, several entity-based spatial index structures such as R-trees [63] are considered state of the art spatial index structures due to their height balance. However, such a tree-based spatial index structure does not perform well in a streaming environment. An R-tree's performance rapidly deteriorates with massive numbers of incremental inserts and deletes due to the constantly required reorganization of the tree. Additionally, the R-tree is designed for disk-based storage instead of main memory as is desired for a DSE. In previous work [92, 123] we used a spatial grid index, based on the grid file [121], for tumbling windows to index sensor observations before interpolation and demonstrated its

ability to efficiently insert, delete, and search for over 250,000 sensor observations. For moving object streams, grid index structures have been used for continuous kNN queries [114], and for tracking flocks consisting of multiple moving objects [163], and to answer object in region queries [108, 110, 106]. Grid indexes have also been used with multi-dimensional data streams and queries, supporting skyline [89] and top-k [113] queries using sliding windows. In [113] both time- and count-based windows are supported with each cell containing a list of tuples sorted by time. Only count-based sliding windows are supported in [89] and a sub-window partitioning of the grid index in introduced to distribute index data to nodes in a cluster for distributed skyline query processing.

## 3.4 Hypothesis and Research Objectives

Our hypothesis is that a stream query operator framework for a DSE utilizing an adaptive k-Shell-based ST-IDW stream operator algorithm, and a Sliding Spatio-Temporal Grid Index (ST grid index) with a shell-based search template transforms massive numbers of asynchronous and mobile sensor observation streams of a continuous ST phenomenon into an ST continuous representation in near real-time while achieving high prediction accuracy of the phenomenon. Specifically, such a DSE stream operator framework achieves a query processing throughput target of 100,000 observation tuples/s to generate a $500 \times 500$ raster representation per query window in 1-3 seconds on a single CPU socket workstation class machine. The hypothesis is verified though evaluation of a prototype implementation of the stream operator framework and assessed with regard to RMSE accuracy and runtime performance using real-world and simulated data streams.

## 3.5 Contributions

To achieve the research objectives, we made the following contributions:

- We introduce the scalable ST Interpolation Stream Query Operator Framework (STI-SQO framework) that transforms a high-throughput stream of sensor

observations into continuous representations in a user-defined way in near real-time. The transformation of the ST point clouds of sensor observations within a query window over the stream into ST snapshots is performed as a stream query utilizing a set of novel operators that are implemented as pipelined, one-pass stream query operators.

- A key component of the STI-SQO framework is an in-memory, Sliding Spatio-Temporal Grid Index (ST grid index) that can be visualized as an ST cuboid, that enables efficient search for proximate sensor observation tuples. This index is used by ST interpolation operators. The ST grid index supports an isotropic view of space and time facilitated by a user-customizable *ST anisotropy ratio*, the introduction of *Isotropic Time Cells (ITCs)*, and a mapping between ITCs and *time blocks* through the use of *time subblocks*.

- Through the introduction of *time blocks* that slice the ST grid index temporally into regularly sized blocks based on the window length and slide, the tuples in the index to be removed or reused can be identified efficiently during sliding window query evaluation. Additionally, using time blocks allows three operations on the ST grid index to be performed concurrently during a window $w_i$ by separate stream operators, namely the operations of removing of old tuples from an expired time block from $w_{i-1}$, inserting new data into a new time block for $w_{i+1}$, and searching the time blocks that constitute $w_i$.

- To retrieve relevant observation tuples from the ST grid index within a specified neighborhood of a grid cell distance in a corrected way based on the anisotropy between space and time, we introduce two *Shell-Based Search Templates*. A shell allows to efficiently determine which index cells are relevant for any raster cell that needs to be interpolated. The shell-based search templates consist of either a single shell used in the Cylindrical Shell Template (CST) or nested shells used in the Nested

Shell Template (NST) to enable a search that iteratively expands in space and time with each shell.

- We introduce a combined ST distance weight for IDW in order to perform Spatio-Temporal Inverse Distance Weighting (ST-IDW) interpolation. Through the use of the ST anisotropy ratio, ST-IDW is able to scale spatial and temporal distance components correctly into a combined ST distance weight. By using ST distance in interpolation, asynchronous sensor observations distributed in time over the window can be used and weighted correctly rather than only the observations at the same time as the Interpolation Center (IC).

- In order to perform ST-IDW, we define two stream-based operator algorithms, ST Shell and ST Adaptive $k$-Shell (ST a$k$-Shell), for generating ST snapshots at each IC. ST Shell uses a user-defined search radius $r$ for each time-space cell and height equal to the length of the window. ST a$k$-Shell is adaptive and searches for tuples in an expanding isotropic sphere using the NST and ITCs and adaptively copes with spatial and temporal data skew.

- The framework supports user-defined query result representations of window queries by allowing them to choose the timestamp within a window for which an ST snapshot is to be constructed (the IC). An ST snapshot generated at the IC represents the predicted state of the phenomenon at that time instant. Instead of a single snapshot per window, the framework also supports a *movie representation* that represents the evolution of a phenomenon within each window as a sequence of raster snapshots and thus captures the temporal continuity of the phenomenon over the window. The movie representation allows the user to specify the *number* of snapshots to be constructed within each window.

## 3.6 Real-time Stream Query Operator Framework for Spatio-Temporal Interpolation

In this section, we give an overview of the proposed ST Interpolation Stream Query Operator Framework (STI-SQO framework).

### 3.6.1 STI-SQO Framework Overview

The STI-SQO framework processes sensor streams in terms of windows over streams. A window is a finite spatio-temporal point cloud of sensor observation. The data of each window is transformed into a spatial raster in near real-time via a novel ST-IDW algorithm. This algorithm is broken up into several stream query operators each of which performs a sub-function. The operators work in a pipelined fashion and those that are computational bottlenecks can be cloned and run in parallel as needed. Thus, the STI-SQO framework is highly scalable and adaptive. Pseudocode for the proposed operators and algorithms can be found in Appendix B.

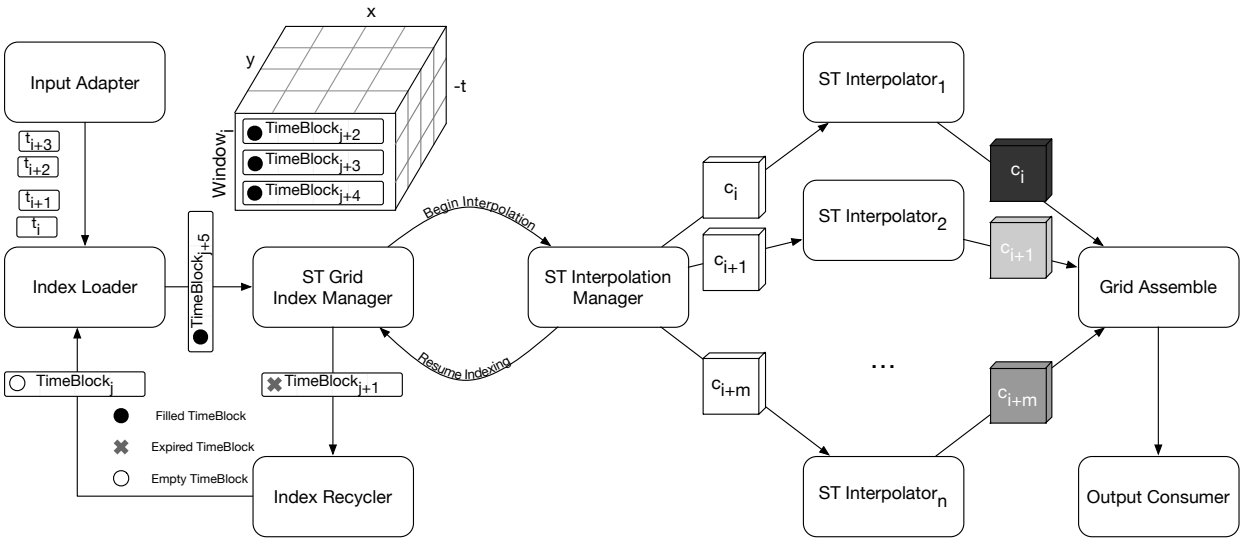



Figure 3.2. ST Interpolation Stream Query Operator Framework operators

The STI-SQO framework consists of the following operators (see Figure 3.2):

- **Index Operators**

  The index operator consists of several, collaborative operators: the *ST Grid Index Manager Operator* maintains the state of the spatio-temporal grid index. It cooperates with the *Index Loader Operator*, which inserts new tuples into the index, and *Index Recycler Operator*, which identifies and clears expired tuples from the index. The index operators are aware of real-time aspects of the grid index, and are designed to insert, search, and delete massive numbers of tuples efficiently.

- **ST Interpolation Manager Operator**

  The ST Interpolation Manager operator coordinates the parallel ST-IDW (Cell) Interpolation Operators and cooperates with the Grid Pane Index Manager Operator with regard to windows and available data in the index.

- **ST Interpolation Operators**

  An individual ST Interpolation Operator instance estimates the value for a single 2D cell of the output raster at a time; it accesses the 3D grid index and searches the index to identify proximate tuples using a search method described in more detail later in Section 3.6.3. It computes the cell value and produces an output tuple. Multiple ST interpolation operators run concurrently and are independent of each other; they share read access to the same grid index.

- **Assemble Operator**

  The *Assemble Operator* sorts and arranges an output raster per window based on the output tuples from the cell interpolation operators.

### 3.6.2 ST Grid Index Supporting Sliding Windows

A core component of STI-SQO framework is the Sliding Spatio-Temporal Grid Index (ST grid index). Tuples arrive in temporal order per stream query window, but they need

to be quickly identified based on their location and time stamp; they need to be indexed as soon as they arrive, and potentially expunged from the index at the end of the current window. In sliding windows, only tuples that are expired need to be purged, while a larger part of the index might be kept for the following window.

#### 3.6.2.1 Index Data Structure

The proposed ST grid index has two spatial (x, y) and one temporal dimension (t). The index can be visualized as a cuboid that slides over the temporal axis, indexing all tuples of the current window of the stream in a 3D ST grid index. The cuboid that is the ST grid index is constructed from a regular spatial grid $sg$ and temporal partitioning $tp$ of the current window of the stream into cells as $sg \times tp = stg$. The $stg$ consists of ST grid cells created from spatial grid cells in $sg$ and temporal time cells of $tp$. A ST grid cell $stgc_{x,y,t}$ is defined as $sg_{xy} \times tp_t$.

With each window slide, the ST cuboid of the ST grid index shifts along the temporal axis towards the future. Tuples no longer within the window are removed and new tuples are inserted into the index. The ST grid index is a data structure shared by all operators in the proposed STI-SQO framework.

#### 3.6.2.2 Temporal Partitioning

Unlike with tumbling windows, where each window contains a unique set of tuples, with sliding windows tuples in the overlapping interval of consecutive windows belong to both windows. There are three regions created from the intersection of consecutive, overlapping sliding windows. These regions are shown for windows $w_{i-1}$ and $w_i$ in Figure 3.3. A subset of tuples in $w_{i-1}$ are also a subset of $w_i$, in $w_{i-1}$ there exist tuples that are too old to be in $w_i$, and in $w_i$ there exist tuples that do not exist in $w_{i-1}$.

In order for the ST grid index to efficiently support overlapping sliding windows, we introduce a temporal partitioning of the ST grid index that allows the ST grid index defined over the overlapping portion of consecutive windows to be reused, expired tuples

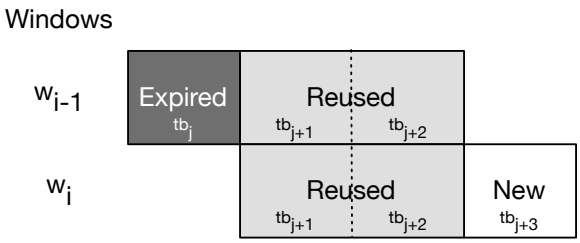



Figure 3.3. Portions of ST grid index that are classified as *Expired*, *Reused*, and *New* for consecutive sliding windows $w_{i-1}$ and $w_i$.

efficiently removed, and the new tuples inserted. To support this function, we introduce *time blocks*. Similar to window panes [86], the temporal length of a time block corresponds to the greatest common divisor of the window length and slide, but for each time block the portion of the ST grid index corresponding to its interval is constructed. Thus, the ST grid index for a window is formed from the union of each time block $tb_j$ belonging to the window $w_i$. Within each time block, a temporal partitioning of *time cells* is used. The partitioning used depends on the query and must have correct time cell locations for all windows that contain each time block. A time block may have one or many time cells within it, and all time blocks for a stream in the query must use an identical time partitioning.

Using time blocks aids in the identification of the portions of the ST grid index that are classified as expired, reused, and new for each window as the same template of time blocks will be used for all windows of the query. For instance, in the example shown in Figure 3.4, the intersection between windows $w_{i-1}$ and $w_i$ corresponds to $tb_{j+1}$ and $tb_{j+2}$ which are *reused* from $q_i$ to $w_{i+1}$. $tb_{j+3}$ is classified as *expired* as it is not in $w_{i+1}$ and thus, removed from the index. $tb_{j+3}$ was not in $w_i$ and belongs to $w_{i+1}$, so the new $tb_{j+3}$ is added to the index.

Since all time blocks have the same time cells and spatial grid, once expired tuples have been removed from a time block, that time block can then be refilled with tuples and again appended to the window.

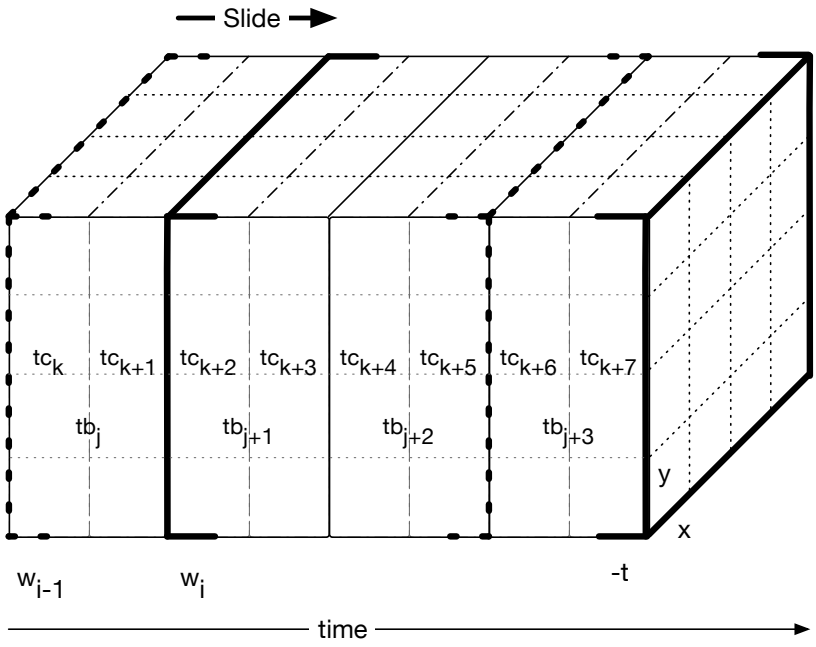



Figure 3.4. Sliding ST grid index structure showing Windows, Time Blocks, Time Cells, and Spatial Grid

#### 3.6.2.3 Index Operations

The ST grid index supports the operations `Insert`, `Delete` and `Search`, which occur concurrently on different parts of the index, and are coordinated by the Index Manager.

**Initialization:** In a single stream query setting, the spatial size of the grid index is likely chosen to be equal to the size of the desired output grid, for simplicity. The number of cells in, and the length of each cell along, the temporal axis in the index depends on the window itself (i.e., range and slide), and whether the temporal dimension is scaled to be isotropic with space through the use of the ST anisotropy ratio $a$. The Index Manager is initialized, which constructs the ST grid index using these parameters. The Index Manager then initializes and configures the Index Loader and Index Recycler.

**Insertion:** The Index Loader operator is responsible for loading tuples into the ST grid index. It inserts tuples in the ST index defined on the time block currently held in its internal state. Tuples are inserted into the grid index based on each tuple's location and time stamp. The Index Loader is aware of window semantics, and able to determine if a

tuple's timestamp belongs in the current window being filled or if it is the first tuple of the next time block. If the timestamp is greater than the end timestamp associated with the time block, the current time block is added to an output queue (input to Index Manager) and a new time is taken from the input queue (output from Index Recycler).

**Search:** The ST cells of the ST grid index are traversed during search by the ST-IDW interpolation operators using their configured search procedure. The ST-IDW operators are notified when all data for the current window are inserted into the index and it is ready to be searched; the ST-IDW operators notify the Index Manager when search of the index is completed but they must also complete within the window time line or be aborted. Searching is described in more detail later on.

**Deletion:** Once search in the index for a window has finished, the Index Manager performs a window slide, outputs the *expired* time block to the Index Recycler, and takes the next *new* time block from the input queue of time blocks from the Index Loader. The Index Recycler processes each time block in its input queue and removes all tuples from all cells so that the time block can later be reused and refilled by the Index Loader.

#### 3.6.2.4 ST Anisotropy Ratio and Adjustment using Isotropic Time Cells

The ST grid index's purpose is to aid in the search for sample points around a prediction location, that is to find tuples that are close in both space and time to a prediction location. However, as different units are used for spatial and temporal measurement and a phenomenon may behave differently in space than in time, distances in space and time are not directly compatible.

Partitioning the ST grid index along the time dimension into time blocks alone yields ST grid cells where the ratio of time units to distance units can only be configured by changing the window configuration or grid resolution. To facilitate an expanding ST search, expansion in time has to be performed in a corrected proportion to expansion in space. This relationship between temporal difference and spatial distance is specified by

the ST anisotropy ratio $a$ (see Equation 3.1) and defines an isotropic mapping between space and time. We use $a$ in order to adjust the temporal size of each ST grid cell without changing the window or grid resolution. These ST grid cells, that now have equal spatial and temporal size with respect to the phenomenon, are termed ITCs.

$$ST\ anisotropy\ ratio = a = \frac{number\ of\ time\ units}{number\ of\ distance\ units} \tag{3.1}$$

In performing ST search by ST interpolation operators, search begins at the IC for the window. Thus, ITCs are defined as a reusable template for isotropic search of the ST grid index. ITCs are introduced to address these issues by providing an isotropic mapping to time and centering this mapping about the IC. With each window slide, the same template of ITCs is applied to each window with the ITC at time index 0 being centered at the IC. Since the alignment of ITCs may overlay differently for each time block in a window, another partitioning of the ST grid index is needed to align the time cells in a time block with ITCs.

In order to align both time blocks and ITCs, we introduce *time subblocks* (see Figure 3.5) as the smallest temporal partition of the grid that can be mapped in multiples to both time blocks and ITCs. Thus, the temporal size of each ST grid cell is the size of a subblock, and time cells and ITCs are two 'views' of the underlying ST grid enabled through a temporal coordinate transformation. A single ITC corresponds to a multiple of subblocks (five subblocks shown in Figure 3.5). On the other hand, once the window slides, data are discarded in units of time blocks, which also correspond to multiples of subblocks (four subblocks in Figure 3.5).

### 3.6.3 ST Index Search Templates

We propose two Shell-Based Search Templates for searching the ST grid index.

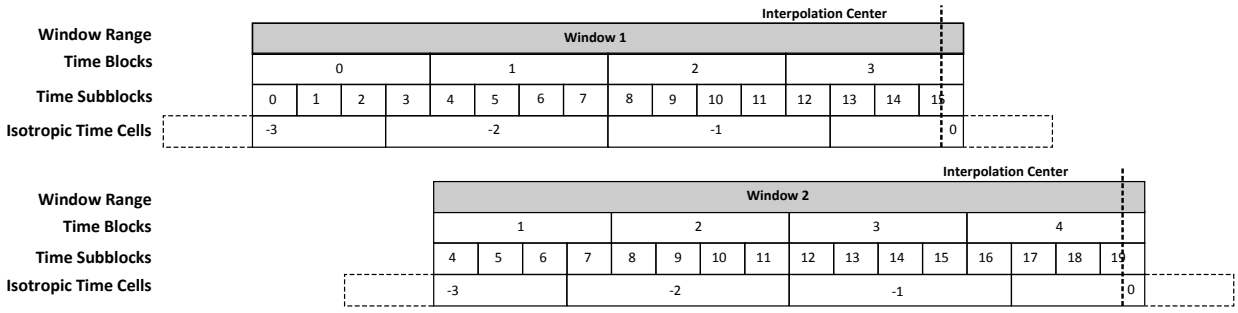



Figure 3.5. Subblocks and Isotropic Time Cells

#### 3.6.3.1 ST Index Search using Shell-Based Search Templates

We introduce Shell-Based Search Templates that are used to search the grid index and select the subset of sample points to use to interpolate a value for a cell. This subset of sample points comes from the grid cells within a neighborhood defined by a so-called shell. A shell consists of array index offsets into the 3D array of the ST grid index for each cell within the defined neighborhood of a cell. Shell-Based Search Templates are reusable, meaning that for a query the same template can be used to search the neighborhood for any cell in the grid, it will work for all windows, and it can be shared between operators accessing the index. Symmetry can be exploited, allowing offsets from the positive ST coordinate cubic space $|x| \times |y| \times |t|$ to be mirrored to the other seven cubic spaces corresponding to combinations of $\pm x \times \pm y \times \pm t$ in the case of ST search

#### 3.6.3.2 Search by ST Proximity with Spatial Flexibility: Cylindrical Shell Template

Using the first Shell-Based Search Template, Cylindrical Shell Template (CST), a interpolation operator searches the ST grid using a cylindrically shaped shell. The CST retrieves all tuples within a cylinder with height equal to the window range and a radius equal to the spatial search radius surrounding each prediction location. All tuples from cells, which have cell centers within this cylinder are returned. Since the CST retrieves tuples from all time blocks in the window, it is not necessary to encode temporal offsets in

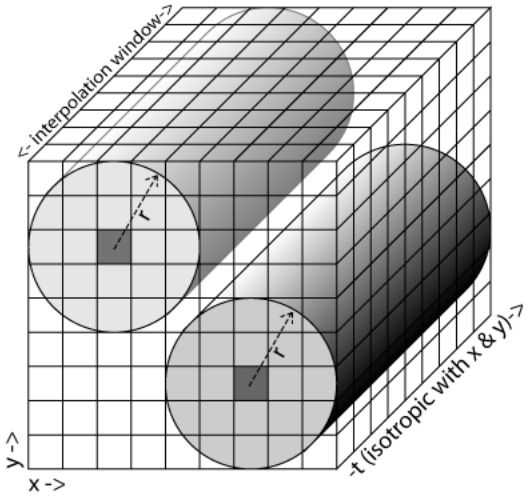



Figure 3.6. Visualization of CST query applied to ST grid index for two cells in the corresponding output grid

the CST. The CST contains a list of the relative spatial offsets for cells within radius $r$ from the northeast quadrant of a Cartesian coordinate system. The coordinates are then mirrored to the other three quadrants. Since a 3D array is used for implementing the ST index, only $x, y$ are needed from the CST as tuples from all time blocks of the current window are returned.

#### 3.6.3.3 Search by ST Proximity with Spatio-Temporal Flexibility: Nested Shell Template

Rather than returning all tuples within a cylinder, the Nested Shell Template (NST) performs an iteratively expanding search in the shape of a sphere. Since expansion is performed iteratively, expansion can be terminated after $k$ tuples are found. The maximum extent of the search of the NST is the same as the CST and bounded by the same cylinder though expansion is performed in the shape of a sphere.

The structure of the NST is a nested list (list of shells) of equidistant spherical shells ordered by distance. Each *shell* consists of coordinate triples with positive $x$, $y$, and $t$ offsets that have equal 3D distance from the origin. Only position coordinate offsets are stored in each shell as the full spherical shell can be constructed through mirroring. The

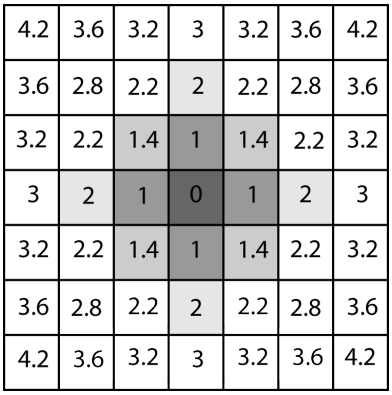



Figure 3.7. NST: NN by cell distance shown in 2D

shells in the NST are ordered by distance. All offsets in the NST are within the spatial radius and expansion is limited to the bounds of the window, thus the NST is constrained by the same cylinder as the CST for a given radius $r$.

The NST requires an $x, y, t$ position to begin the search and terminates expansion of additional shells once $k$ tuples have been found or all cells within the search cylinder have been visited, whichever occurs first. The NST is constructed by generating all combinations of coordinates that fall within the search radius and time window and sorting the list based on the cells' distance to the center of the cell being interpolated.

### 3.6.4 ST-IDW using Shell-Based Search Templates

We first discuss stream-based ST-IDW and present two algorithms for performing stream-based ST-IDW.

#### 3.6.4.1 Approximating Continuous Phenomena using an ST Interpolation Center

The problem of representing spatial snapshots of a continuous phenomenon over space at a certain time instant is well established in traditional GIS. This is performed under the assumption that all sensor observations are collected at exactly the same time stamp by all sensors. When the assumption of synchronous updates is removed, the temporal location of

each sample point within a window must be considered when representing the phenomenon, and requires the use of ST interpolation.

In order to accurately represent the spatio-temporal development of a continuous phenomenon, we introduce the concept of the *IC*. The IC is the time within each window at which a spatial snapshot is constructed using ST interpolation; it represents the predicted state of the phenomenon at that instant within the window based on samples that are in spatial and temporal proximity. An IC, with time $t_{ic}$, is calculated as a relative integer time offset from the start of the window. An IC offset lies within the interval $t_s \leq t_{ic} < t_e$. The integer time offset operates identically to a linear array. A window of length 5 time units maps to an array with length 5 and indices ranging from 0 to 4. Specifying an IC at offset 2 maps to the midpoint of the window and results in a snapshot at 2 ½ time units and 4 maps to the last cell in the window and results in a snapshot at 4 ½ time units.

We choose not to pre-define a fixed IC, and allows users to specify an IC based on their needs. This supports different user and application needs including near real-time phenomenon representation using an IC near the end of the window or a summary of the phenomenon using an IC in the middle of the window.

#### 3.6.4.2 Stream-Based ST-IDW

ST interpolation is performed in parallel by ST-IDW operator instance clones, which produce an ST snapshot for each IC. The operator clones are independent of each other and share read access to the same ST grid index. Each ST-IDW operator takes a tuple representing the grid cell to interpolate, calculates an interpolated value using ST-IDW, and outputs the tuple with the interpolated value. The interpolated values from all ST-IDW operators in the query are assembled into the output grid by the Grid Assemble operator as they arrive in its input queue. The interpolation of each cell in the prediction grid is completely independent from every other cell, and since interpolating the entire raster is the most time consuming step in the query plan, running ST-IDW operators in

parallel with one operator per thread reduces the computation time. For this type of simple-to-parallelize problem, the resulting speedup factor compared to using only a single thread approaches the number of threads, as long as the number of threads is less than the number of threads that can be executed concurrently by the CPU.

ST-IDW performs ST interpolation by performing IDW and replacing the spatial distance weight with a combined ST distance weight calculated using Equation 3.2. In Equation 3.2 the spatial distance between a prediction location and a tuple is calculated in terms of great-circle distance using the haversine formula [151]. The temporal distance is calculated from the difference between the interpolation center and the timestamp of the tuple and is scaled by the anisotropy ratio $a$ (see Equation 3.1) and length of the grid diagonal in both arc degrees and number of spatial grid cells in order to combine the spatial and temporal distance components into a combined ST distance.

$$d(cell_c, tuple_t) = \sqrt{hav(cell_c, tuple_t)^2 + \left(\frac{grid\ diag\ in\ arcdegrees}{grid\ diag\ in\ cells} \times \frac{t_{ic} - t_t}{a}\right)^2} \tag{3.2}$$

Parameters to ST-IDW additionally include a power factor, a variable spatial search radius around a cell ($r$), and, in some cases, a threshold level of $k$ sample points to use to interpolate each grid cell.

#### 3.6.4.3 ST-IDW Interpolation using ST Shell Approach

In the ST Shell to ST-IDW approach, the CST is used to access the ST grid index using a ST cylindrical template of spatial radius $r$ around each prediction location. All sample points found using the CST are weighted by their combined ST distance (Equation 3.2) using ST-IDW.

The ST Shell approach performs well if the search radius $r$ is small with a given window range, but as the radius increases, the number of grid cells that need to be accessed and the number of tuples to be processed increases significantly. In cases when samples are sparse, it might be necessary to require a large preset spatial search radius until sufficient

sample data are found, and even simple checking of cells for tuples is expensive. However, if tuples are abundant, a small search radius is sufficient.

#### 3.6.4.4 ST-IDW using ST Adaptive $k$-Shell Interpolation Algorithm

Based on the previous discussion of the ST Shell approach, using a constant interpolation radius for all prediction grid cells can become inefficient for sparse and skewed data sets. Thus, we introduce the ST a$k$-Shell algorithm implementing ST-IDW using the NST access method. The ST a$k$-Shell adapts to the available data distribution automatically.

Since ST-IDW assigns more weight to samples closer in space and time than to those farther away, samples at greater distance contribute statistically less information about the value of the cell being predicted. At some point, any further samples can be disregarded without affecting the error of the interpolation beyond some tolerance. We exploit this and propose the ST a$k$-Shell approach to ST-IDW interpolation. As described above, for each cell in the output raster, tuples from successively more distant cells (distant in both time and space) are used to determine the predicted value, and when the specified number of tuples have been processed the operator terminates interpolation for this cell.

The ST a$k$-Shell approach works well in regions where data are dense as once $k$ tuples are found, the value of the cell in the prediction grid is calculated. An increase in the search radius $r$ means that the algorithms can search farther in space and time in areas where data are sparse. Increasing radius from an initial radius $r_0$ to a larger radius $r_1$ will increase time to interpolate every cell in the prediction grid where there are less than $k$ tuples within $r_0$. Cells having $k$ tuples within $r_0$ will experience no change in interpolation time using $r_1$.

## 3.7 User-Defined Stream Query Support

Creating a stream based ST phenomena depends often on the application, which influences appropriate parameter setting. Thus, these parameters need to be user-defined and controlled by the user. The STI-SQO framework allows the users to configure the STI-SQO framework to their requirements, discussed as follows.

### 3.7.1 Spatio-Temporal Interpolation Center

A dynamic ST phenomenon is represented as a raster that represents the phenomenon as a ST snapshot capturing the predicted state of the phenomenon at the timestamp of the IC using the sample points from the window. Since the IC is specified by the user, it allows the users to specify the IC that is best for their requirements.

An IC, $t_{ic}$, defined at the end of the query window, $t_e$, predicts the latest state of the phenomenon and results in a time decaying effect of earlier samples collected at the beginning of the window. This means that the values at the beginning of the window have less influence than the ones towards the end of the window.

For applications that are focused on continuous monitoring or archiving dynamic phenomena, choosing an IC in the middle of the window, i.e., $(t_s + t_e)/2$, provides the best summary of the phenomenon's changes during the window, as it considers both samples before and after $t_{ic}$ (within the window) for interpolation. While the end of window and the middle of window are the two most likely ICs, there are no restrictions on when to place the IC within the query window, only that it must be at an integer time offset.

It is advisable that a user chooses an IC in the interval between the middle and end of the window based on their prioritization of minimizing latency between the current time and snapshot time (IC at the end of the window) and maximizing accuracy (IC at the center of the window). Thus, using an IC near the end of the window is best for real-time phenomenon representation, while for maximizing accuracy and best summarizing the phenomenon an IC at the center of the window is best.

Using the IC approach, we distinguish the following window query result types over continuous phenomenon: *ST snapshot stream queries*, which return a stream query results consisting of a single ST snapshot per window, and *ST movie stream queries*, which return a series of snapshots and thus captures the evolution of the phenomenon over a window.

#### 3.7.1.1 ST Movie Stream Queries

A user may be interested in capturing the temporal variation of a phenomenon over a window, which is not captured by a single ST snapshot. An intuitive way to capture this variation is a 'movie' of the phenomenon, i.e., a series of ST snapshots, or frames, within a window. Each frame of the movie is a spatial raster produced using ST interpolation. However, the input for each interpolation is smaller, i.e., a fraction of the window size. For example, a 5 minute window can be represented at different frame rates such as one frame every 30 seconds (10 frames overall), or one every minute (5 frames overall). Each frame can be computed using time decaying ICs at the end or middle of the window subinterval, depending on the application needs. Movie queries are described and tested in [165] in more detail.

### 3.7.2 Spatio-Temporal Anisotropy

The isotropic view of the ST grid index using ITCs and ST distance calculation can be adjusted by the ST anisotropy ratio based on the ST anisotropy of the phenomenon. The ST anisotropy ratio depends on the spatial and temporal units used to observe the phenomenon and how the phenomenon evolves over space and in time. As individual phenomena may behave in a different manner and may be observed using various spatial and temporal units, the ST anisotropy is provided as a parameter for the user to configure the STI-SQO framework based on the behavior of the phenomenon and spatial and temporal units.

### 3.7.3 ST-IDW Parameters

Several parameters exist to configure the behavior of the two algorithms for ST-IDW, ST Shell and ST a$k$-Shell. These are the radius $r$, number of sample points $k$, and the power $p$. The parameters $r$ and $p$ are required for both ST Shell and ST a$k$-Shell and $k$ is used only by ST a$k$-Shell. With $r$ adjusting the spatial radius of the search cylinder, increasing $r$ widens the search for sample points, but sample points farther away may not be as closely correlated as closer points. Thus, $p$ adjusts how the weight of points based on distance should decay.

In order to adapt to the skew of distribution of observations over space and time in the window, $k$ is used to terminate the search once $k$ tuples are found. Using $k$ is advantageous as areas with sparse sample points are allowed to expand to the full radius $r$ while in areas where the sample points are dense the search stops once $k$ sample points are found. Thus, by using $k$ the ST a$k$-Shell operator is able to adapt to data density, with $k$ being the termination condition when data are dense and $r$ for when data are sparse.

### 3.7.4 Additional Parameters to the STI-SQO Framework

There are additional parameters available for the user to configure including the width and height of the grid used for the ST snapshots, the length and slide of the window, and the number of ST-IDW operator clones.

## 3.8 Summary

In this chapter we presented the STI-SQO framework and its main components. The STI-SQO framework is designed to process high-throughput sensor observation streams in near real-time and to produce ST snapshots in the form of rasters using ST-IDW. This is performed using a stream query operator framework including an ST grid index, consisting of the Index Manager, Index Loader, and Index Recycler sub-operators, and parallel ST interpolation operator clones that interpolate the raster cell-by-cell using ST-IDW. The ST

Shell Approach and ST Adaptive $k$-Shell algorithms for ST-IDW are presented along with their corresponding ST grid index shell-based search templates, Cylindrical Shell Template and Nested Shell Template.

In the following chapter, we will discuss the performance evaluation that includes the discussion of the test data sets, a first set of tests that investigates optimal parameter choices based on RMSE evaluation and secondly, throughput performance tests for these selected parameters.

# CHAPTER 4
# ACCURACY AND RUNTIME PERFORMANCE EVALUATION OF STI-SQO FRAMEWORK

## 4.1 Introduction

This chapter continues with the assessment of the STI-SQO framework introduced in Chapter 3. We introduce relevant test data sets and discuss parameter configurations. The first set of experiments analyzes the impact of parameter choices on RMSE; the second set of experiments evaluates the throughput and runtime for the configurations with the lowest RMSE. While the STI-SQO framework supports many configurations, we selected several scenarios that are representative of massive mobile sensors deployed over small- and large-scale street networks and dynamic ST phenomena.

## 4.2 Experimental Setup

Since sensor data streams in high spatial and temporal density of continuous ST phenomena are not available (yet) for the STI-SQO framework, we built an agent-based model to simulate the generations of such massive sensor data streams for different test environments. Using the simulation, simulated sensor agents sampled a model of a real world spatio-temporal event of radiation deposition (Fukushima 2011). The observation streams from these simulated sensors were used as input to evaluate the approach with respect to accuracy of prediction and runtime for different parameter configurations.

### 4.2.1 Radiation Data Sets

The ST phenomenon we considered represents the 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 [172] and SPEEDI [34] using

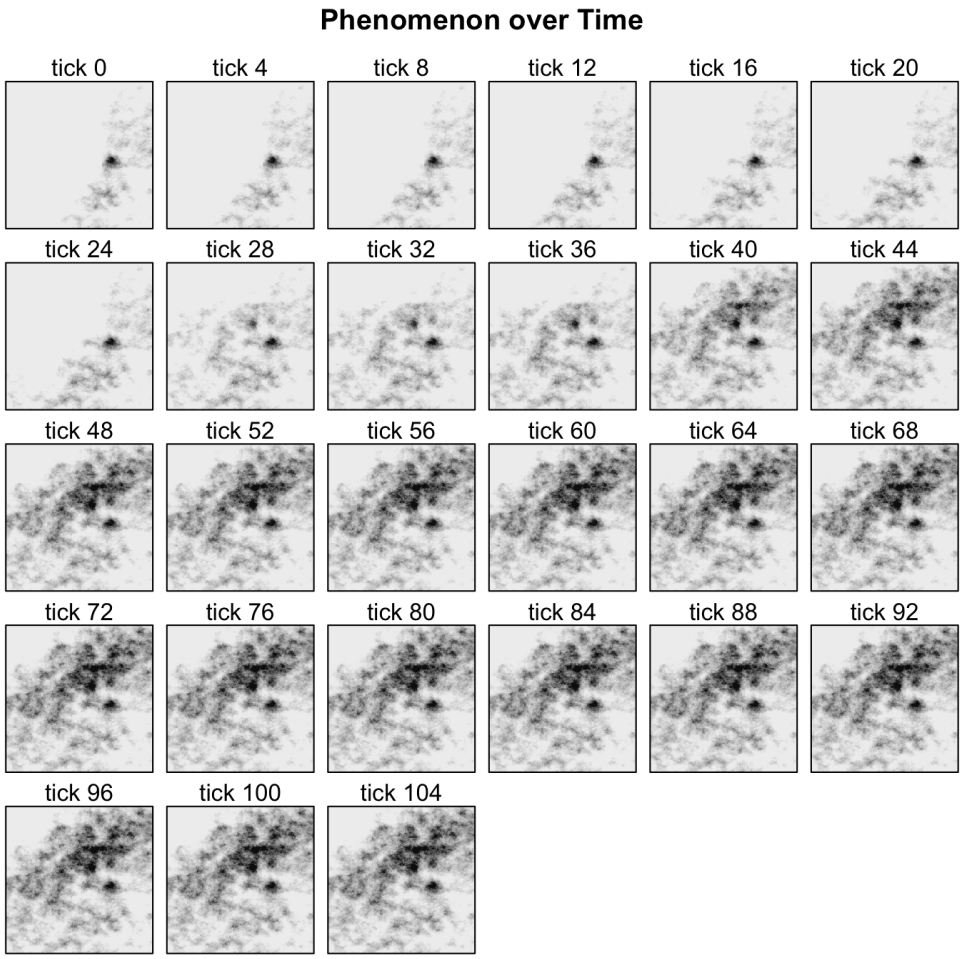



Figure 4.1. Evolution of phenomenon over time

the process described in [92]. From those snapshots, a sequence of snapshots of the event between the fourth and fifth day after the disaster in 15-minute sampling increments was generated, a subset of which are shown in Figure 4.1. For the snapshots, tick 0 maps to March 15 00:00 UTC, 9:00 JST and tick 104 is March 16 02:00 UTC, 11:00 JST. The time interval during which the phenomenon is highly dynamic occurs before tick 40, which corresponds to March 15 10:00 UTC, 19:00 JST, and after that time the phenomenon exhibits little change.

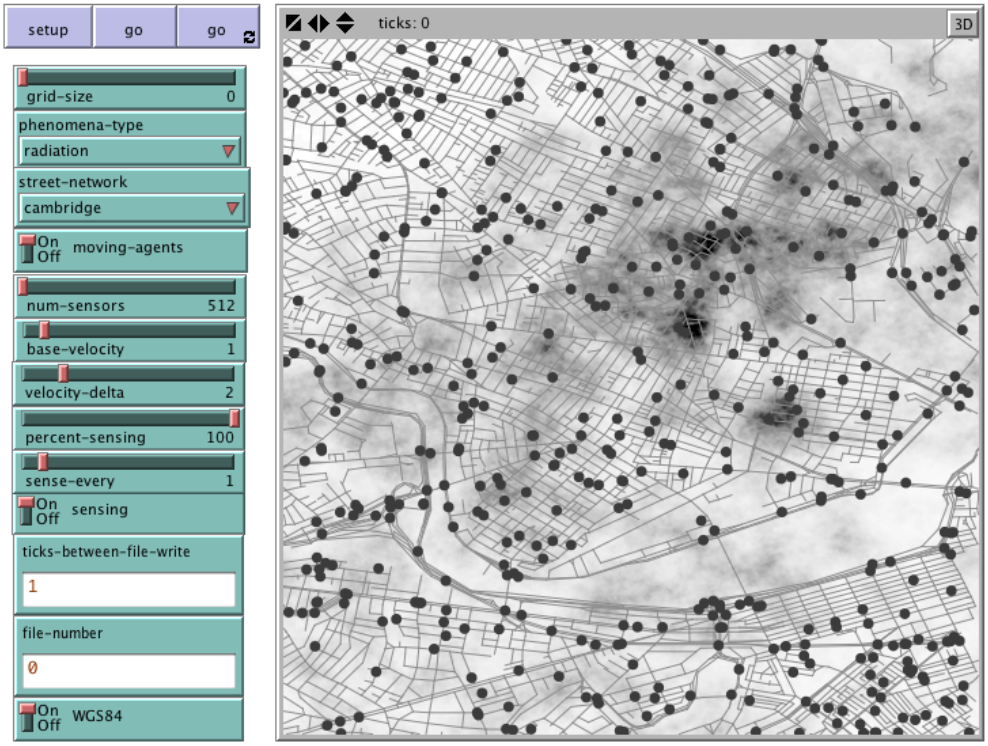
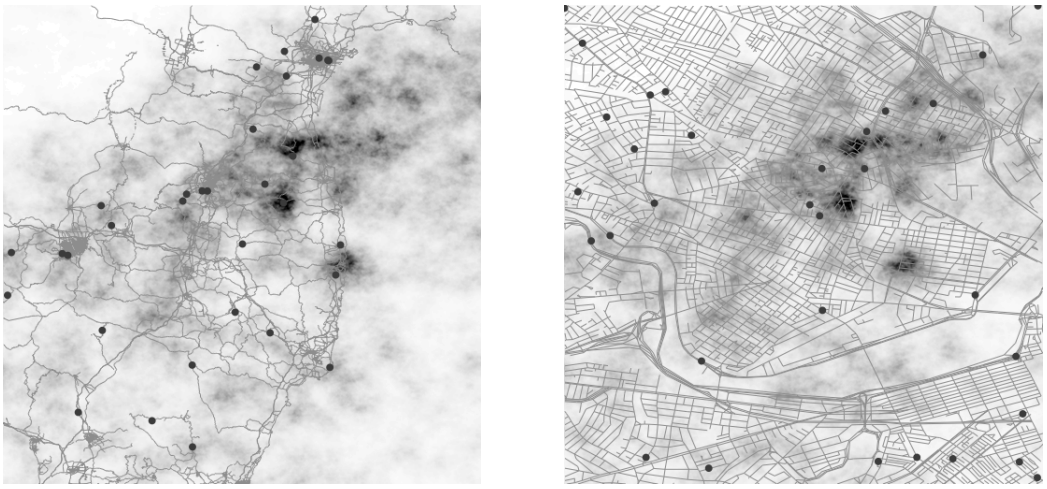



Figure 4.2. Data generator built in NetLogo

### 4.2.2 Stream Data Generator

To generate high-density data streams, a *data generator* simulation (shown in Figure 4.2) of moving sensor agents sampling a ST phenomenon in their environment and creating sensor streams was implemented in NetLogo [166]. The data generator uses an an ST phenomenon input that is represented as a sequence of raster snapshots.

The data generator sampled the phenomena snapshots using 2M ($2^{21}$) sensor agents that moved as the model iterated through the phenomenon snapshots. The mobility of sensors was simulated by movement constrained to links in a street network and an unconstrained random walk. In our test cases, the street networks, shown in Figure 4.3, are a) a large region of Japan with a coarse road network and b) a small area of the Cambridge and Boston, MA area with a dense road network. Additionally, three random walk data sets were generated, each by a different initialization of the model.



(a) Fukushima, Japan (b) Cambridge, Massachusetts

Figure 4.3. Radiation plume and simulated sensors over (a) Fukushima, Japan and (b) Cambridge, Massachusetts

In order to assess the spatial skew of points in the generated Cambridge, Japan, and Random simulated mobile sensor data sets, the distributions of distance from each sensor to every other sensor were generated. A subset consisting of 8K (8,192) sensor observations from each of the street network data sets was used to generate the distributions. The plot (see Figure 4.4) shows that the distribution of sensors in the Random and Cambridge data sets are very similar while the distribution for the Japan data set is skewed towards shorter distances, likely due to a relatively large portion of the image constituting water where no sensors were deployed.

### 4.2.3 Implementation and Run-time Environment

The STI-SQO framework was implemented in Java as a DSE prototype consisting of pipelined operators connected by queues designed for testing the performance of the ST grid index and ST-IDW (implementations of ST Shell and ST a$k$-Shell) for varying data sets and interpolation parameters. We did not consider any of the other DSE components, e.g., load shedding. The Spatial4j [94] library was used for calculating spatial distance using the haversine formula.

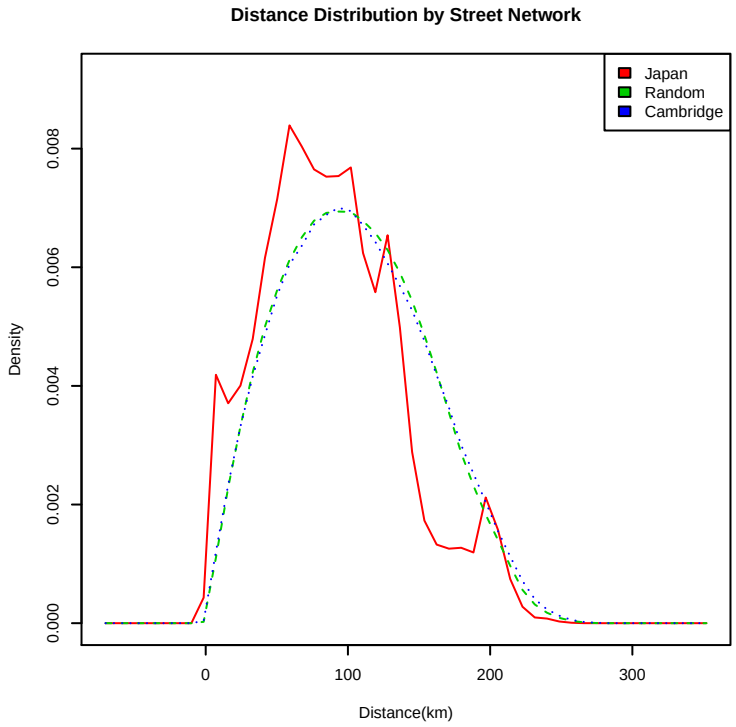



Figure 4.4. Distance distribution by street network

The experiments were run on an HP Z440 workstation with a 3.5 GHz, six core (Intel Xeon E5-1650v3) processor and 32 GB of DDR3 1866 memory. The system ran Ubuntu 14.04 and Java 1.8.0_131. The Java heap size was set at 16GB, which was used to buffer the input stream and to run all operators.

### 4.2.4 RMSE Validation

In order to evaluate the accuracy of the output rasters produced during experiments, the Root Mean Square Error (RMSE) is used to quantify the error between each output raster, $p$, and the ground truth raster (input to the data generator), $o$, for the corresponding time stamp and is shown in Equation 4.1.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} \left(p_{t,i} - o_{t,i}\right)^2}{n}} \tag{4.1}$$

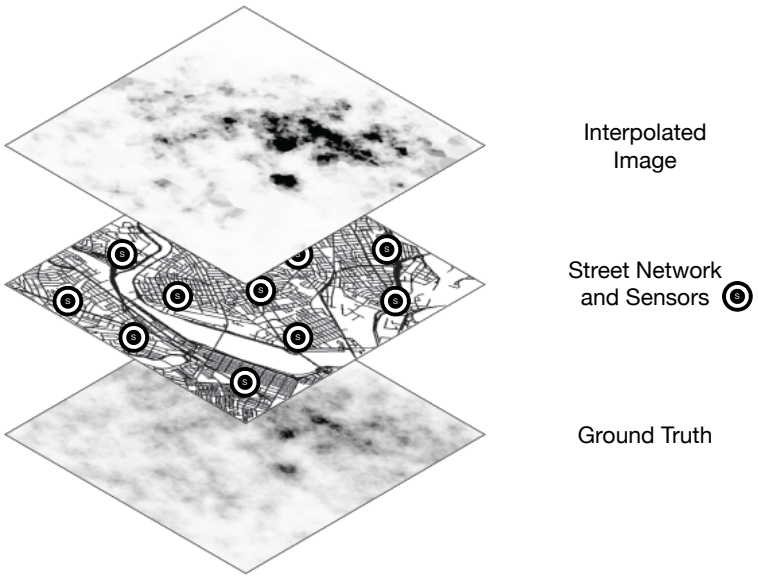



Figure 4.5. Comparison between interpolated image and ground truth

For a predicted output raster of $p$ at timestamp $t$, where $t = WindowStart + IC$ having $n$ raster cells and the corresponding ground truth $o$ for the same timestamp $t$, the RMSE is calculated from the difference between each corresponding cell location $t, i$.

To make the RMSE comparable among different snapshot timestamps, the RMSE was normalized by dividing the RMSE by the mean of $o_t$ to produce NRMSE (see Equation 4.2.) In essence, the NRMSE captures the RMSE as a proportion of the mean of the ground truth.

$$NRMSE = \frac{RMSE}{\bar{o_t}} \tag{4.2}$$

### 4.2.5 Runtime Testing

Runtime tests were performed running each framework query operator with its own thread, except for the ST-IDW operator where 8 clones were run. For the runtime tests, results were averaged for 5 iterations. Runtime was calculated as:

$$runtime = mean(t_{index}) + mean\big(max(t_{idw_1}), \ldots, max(t_{idw_n})\big) \tag{4.3}$$

## 4.3 RMSE Performance Evaluation

In this section the performance of ST Shell and ST a$k$-Shell was evaluated with respect to RMSE. The RMSE tests aim to identify the configurations with the lowest RMSE to be evaluated further in the runtime tests in Section 4.4 and RMSE vs. runtime in Section 4.5.1.

For the following tests, a sliding window with a length of 32 ticks (time units) and a slide of 4 ticks was used unless noted otherwise. Tests were configured to assess the impact of the ST anisotropy ratio $a$, radius $r$, $k$, IC location within a window, timestamp of IC, rate of phenomenon change, the number of sensors $n$, and the width of the window. A random sample of $1/32$ of the sensor observations taken for each time tick to simulate the stream of asynchronous sensor observations. The tests were configured to produce a $500 \times 500$ raster grid with each window slide. For plots, *snapshot time* is defined as the time tick at which a snapshot is created, specified by $WindowStart + IC$.

### 4.3.1 ST Anisotropy Ratio

The first set of ST tests investigated the impact of the ST anisotropy. The tests were performed using an IC located at the center and end of the window for both the ST Shell and ST a$k$-Shell algorithms. Results are presented for ST Shell, using $r = 32$ for an IC located at the center (see Figure 4.6) and end (see Figure 4.7), and for ST a$k$-Shell, using $r = 32$ and $k = 8$ for IC center (see Figure 4.8) and end (see Figure 4.9). These results show the NRMSE for the time tick at which each snapshot is created. Results are shown for $p = 4$, and additional combinations of $p$ and $a$ are presented in Section 4.3.2. The plots are presented separately by interpolation algorithm (ST Shell and ST a$k$-Shell) and IC, series and each plot shows the street network (Japan, Cambridge, and Random). From these plots, the following anisotropy ratios were eliminated from future testing: 0.0625, 0.125, 0.25, 0.5, 16, 32 because they resulted in higher NRMSE compared to other ST anisotropy ratios tested. ST anisotropy ratios $a = 16$ and $a = 32$ had the highest NRMSE

before t=50 using an IC at the end of the window and similarly before t=46 when using an IC at the center of the window. When $a = 16$ and $a = 32$ did not have the highest NRMSE, $a \in \{0.0625, 0.125, 0.25, 0.5\}$ had the highest NRMSE. Further analysis to identify the optimal value of $a$ is performed in Section 4.3.2 using the remaining $a \in \{1, 2, 4, 8\}$.

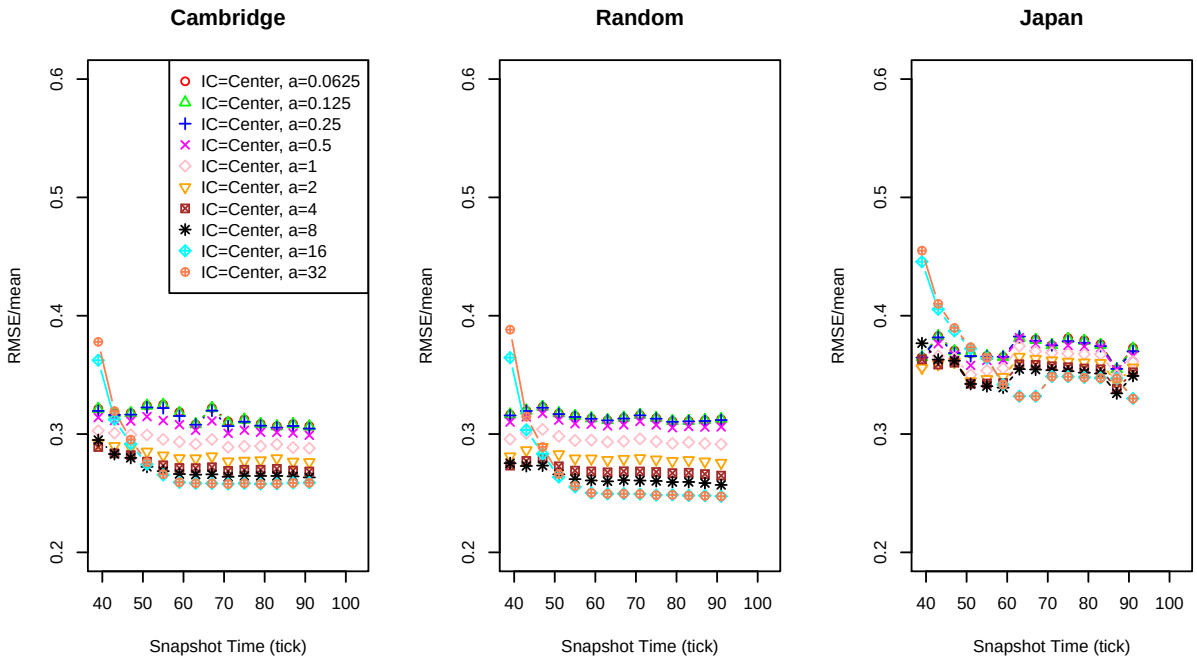



Figure 4.6. ST anisotropy RMSE using ST Shell, $r = 32$ $p = 4$, $n = 128$K sensors, and an IC at the center of the window

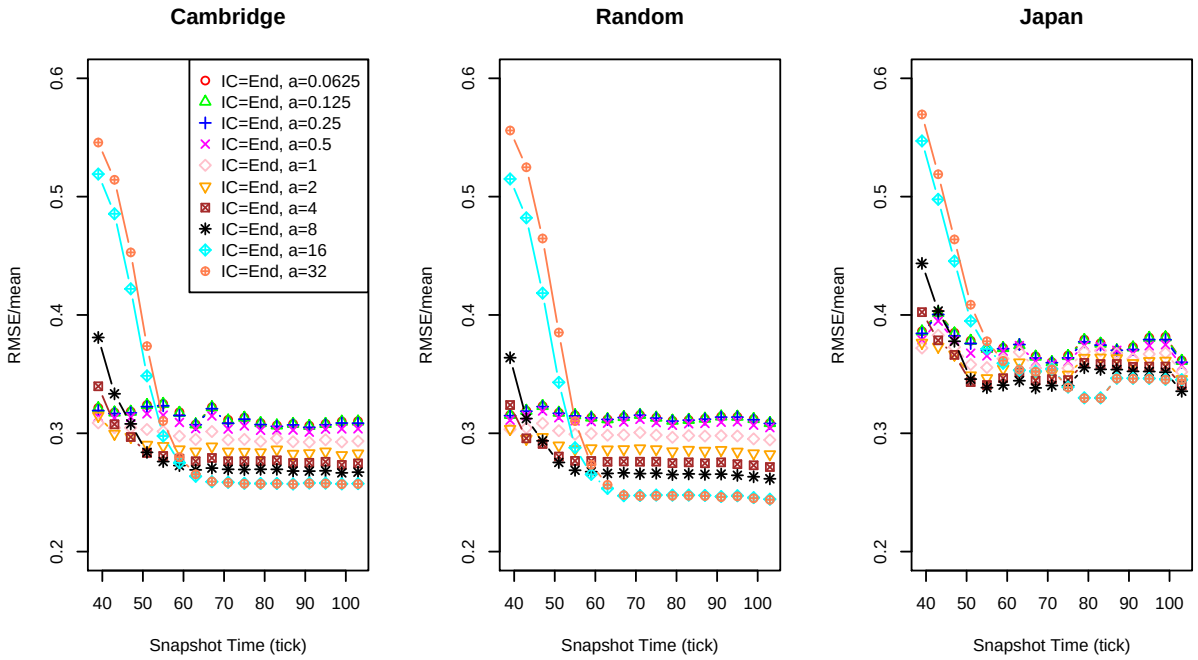



Figure 4.7. ST anisotropy RMSE using ST Shell, $r = 32$ $p = 4$, $n = 128$K sensors, and an IC at the end of the window

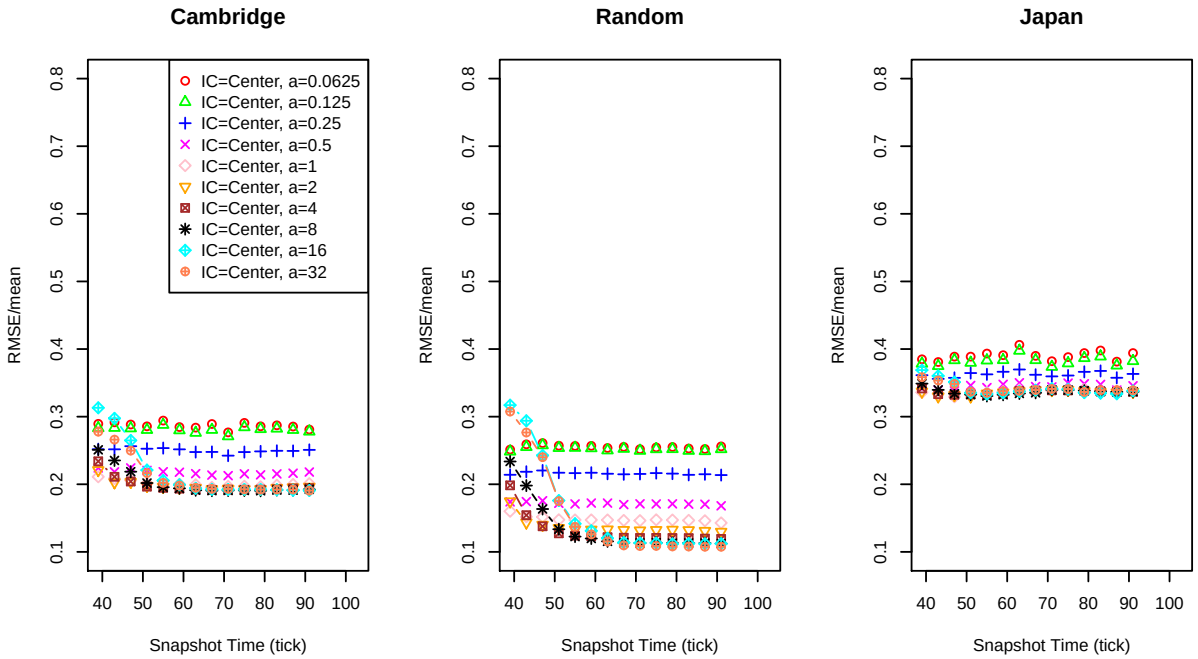



Figure 4.8. ST anisotropy RMSE using ST a$k$-Shell, $r = 32$, $k = 8$, $p = 4$, $n = 128$K sensors, and an IC at the center of the window

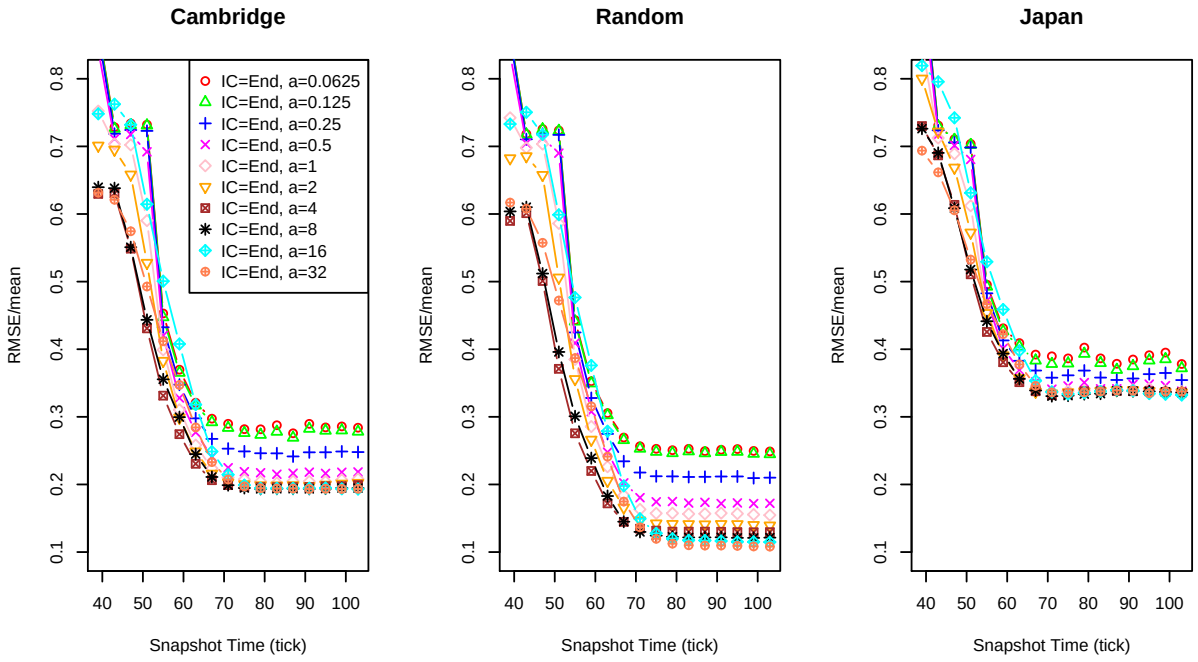



Figure 4.9. ST anisotropy RMSE using ST a$k$-Shell, $r = 32$, $k = 8$, $p = 4$, $n = 128$K sensors, and an IC at the end of the window

### 4.3.2 ST-IDW Power

Further analysis was performed to investigate the best combination of $a$ and the ST-IDW power $p$ using $a \in \{1, 2, 4, 8\}$ and $p \in \{1, 2, 3, 4\}$. From the results, presented in Figures 4.10, 4.11, 4.12, and 4.13, the combination of $p = 4$ and $a = 4$ was chosen as the configuration to use for the remaining tests. For an IC at the end of the window, the combination $p = 4$ and $a = 4$ resulted in the lowest NRMSE for the Cambridge, Random, and Japan data sets in Figures 4.10, and 4.11.

With an IC at the center of the window, using $a = 4$ (see Figure 4.12) indicated that $p \in \{3, 4\}$ were the two powers that resulted in the lowest NRMSE with both having very similar graphs. Since $p = 4$ performed similarly to $p = 3$ and was clearly the best option for an IC at the end of the window, $p = 4$ was chosen as the power to use for all tests. Looking at the NRMSE for an IC at the center of the window and using $p = 4$ (see Figure 4.13), $a = 4$ the lowest NRMSE second to $a = 4$ after tick 54 when the phenomenon is static and

$a = 4$ has a lower NRMSE than $a = 8$ before tick 54 when the phenomenon is changing. Thus, the parameters $p = 4$ and $a = 4$ are selected for the remaining tests.

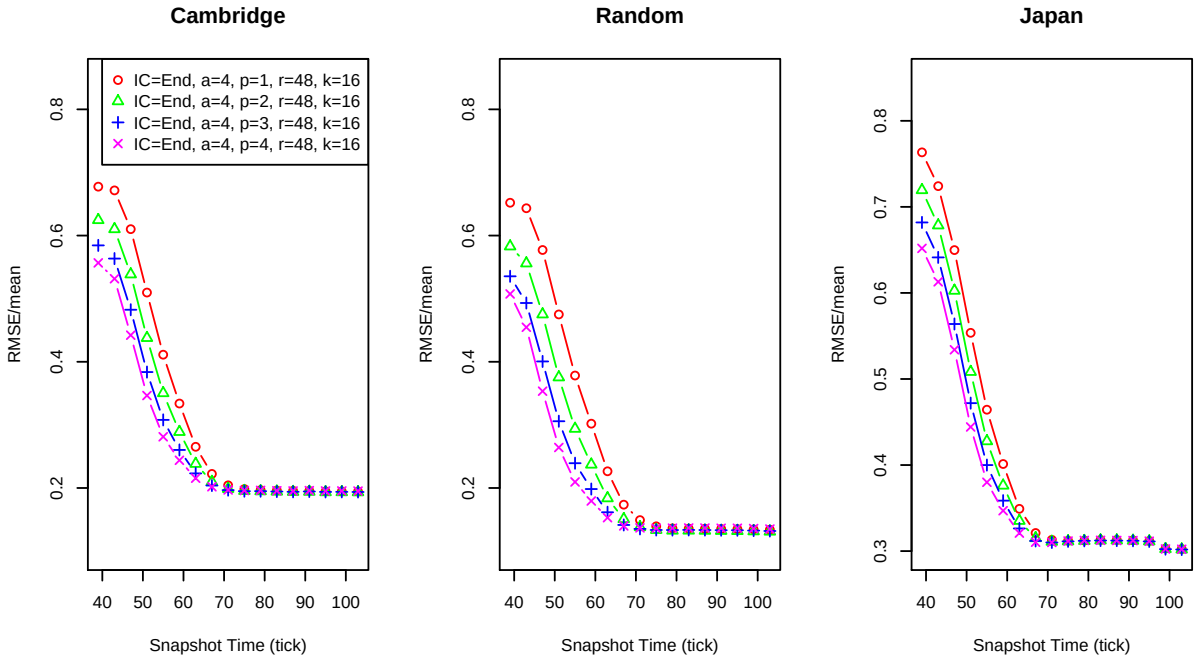



Figure 4.10. ST a$k$-Shell NRMSE by power using $r = 48$, $k = 16$, 128K sensors, an IC located at the end of the window, and ST anisotropy ratio $a = 4$

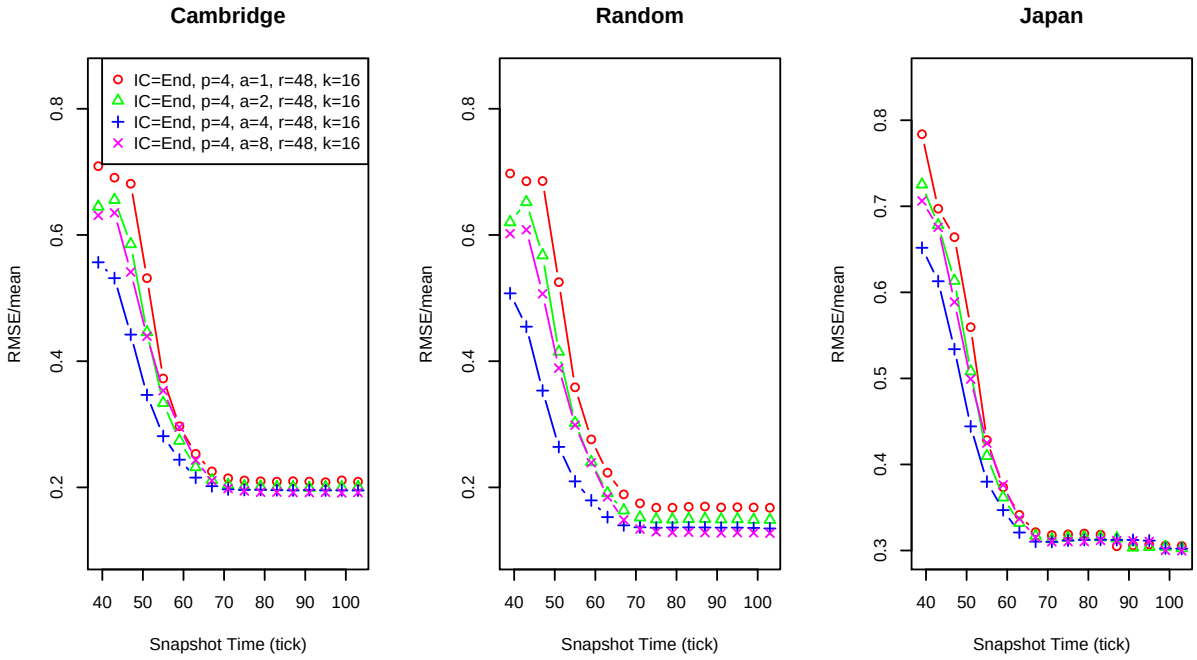



Figure 4.11. ST a$k$-Shell NRMSE by ST anisotropy ratio using $r = 48$, $k = 16$, 128K sensors, an IC located at the end of the window, and power $p = 4$

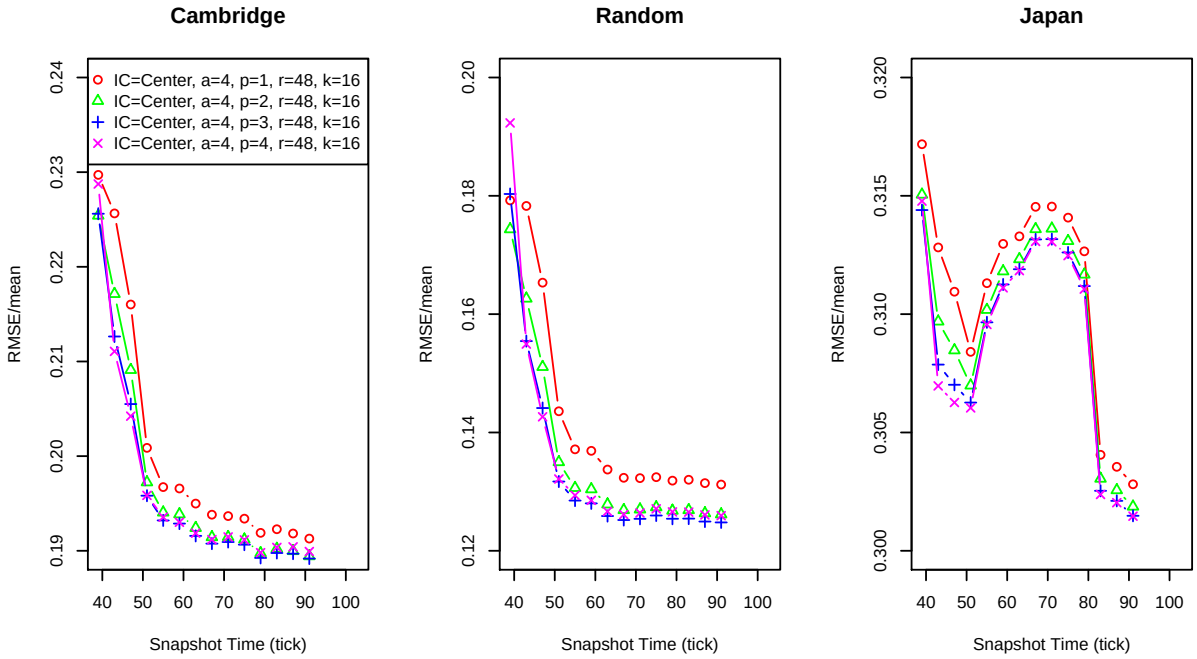



Figure 4.12. ST a$k$-Shell NRMSE by power using $r = 48$, $k = 16$, 128K sensors, an IC located at the center of the window, and ST anisotropy ratio $a = 4$

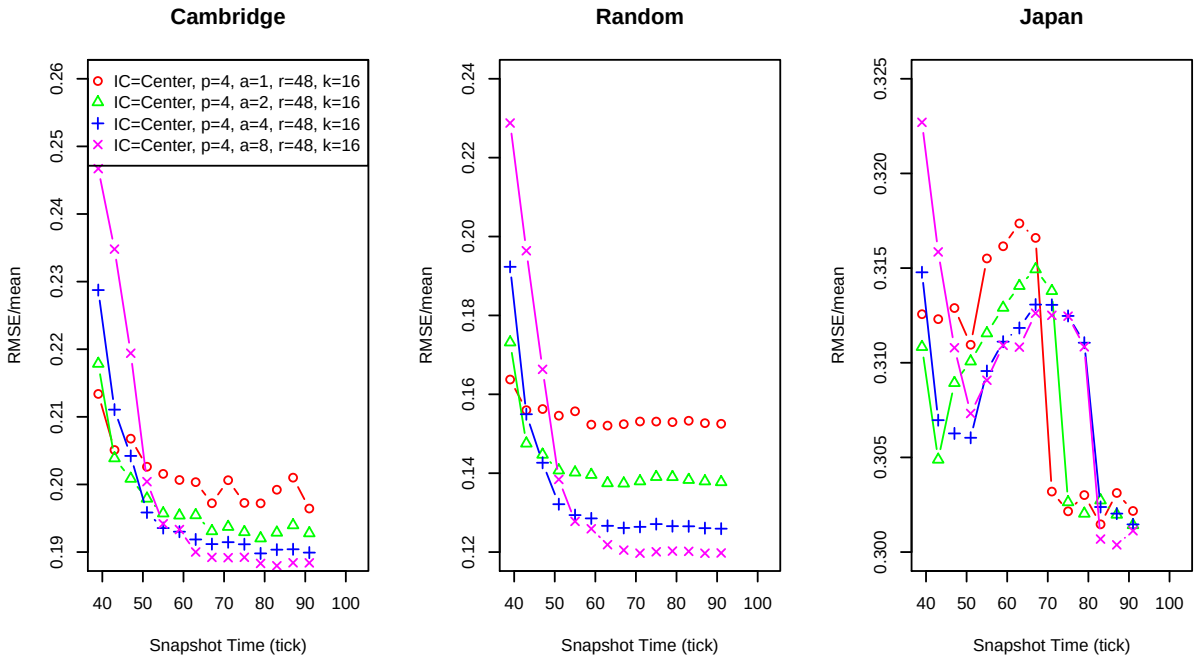



Figure 4.13. ST a$k$-Shell NRMSE by ST anisotropy ratio using $r = 48$, $k = 16$, 128K sensors, an IC located at the center of the window, and power $p = 4$

### 4.3.3 ST Shell Approach: NRMSE by Radius

For ST Shell radii $r \in \{16, 24, 32, 40, 48\}$ are evaluated for RMSE shown in Figure 4.14. The results show that the smallest radius tested, $r = 16$, performed best for the dense Cambridge street network and random walk data sets while for the more sparse Japan street network the larger radii $r \in \{40, 48\}$ were the two configurations with the lowest NRMSEs. Previous work [165] showed that $r = 32$ was able to interpolate nearly all the entire land area portion of the Japan data set study area while smaller radii tested were not.

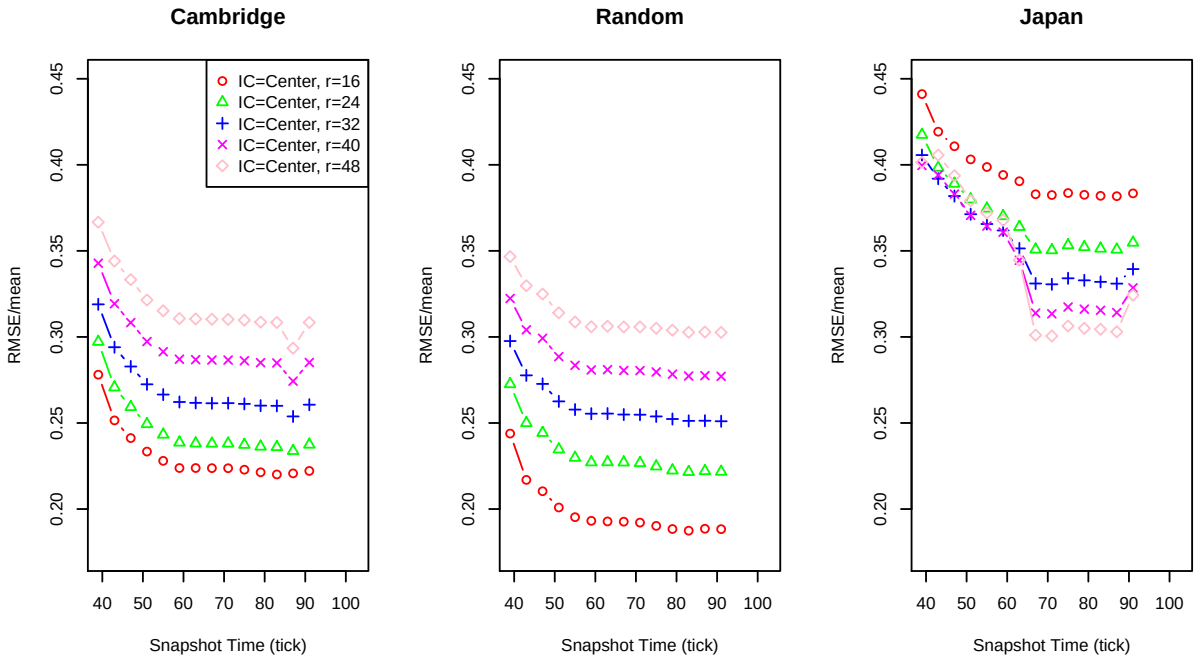



Figure 4.14. ST Shell NRMSE using 128K sensors and an IC located at the center of the window

### 4.3.4 ST Adaptive $k$-Shell: NRMSE by k

Using a radius $r = 48$, we test the impact of $k \in \{1, 2, 4, 6, 8, 12, 16\}$ on RMSE. The results show that increasing $k$ resulted in a lower RMSE. Additionally, as the number of sensors increase, from 16K to 128K, using larger values of $k$ result in further decrease in RMSE. Results are presented for 16K (Figure 4.15) and 128K (Figure 4.16) sensors with an IC at the end of the window.

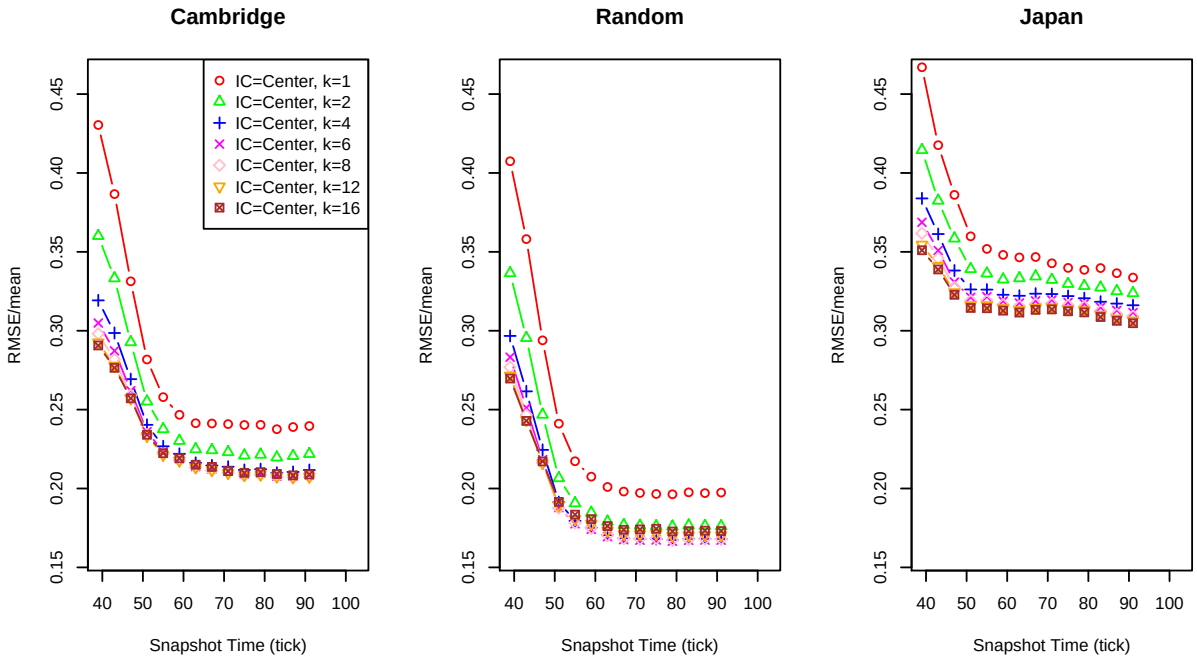



Figure 4.15. ST a$k$-Shell RMSE using $r = 48$ and 16K sensors and an IC located at the center of the window

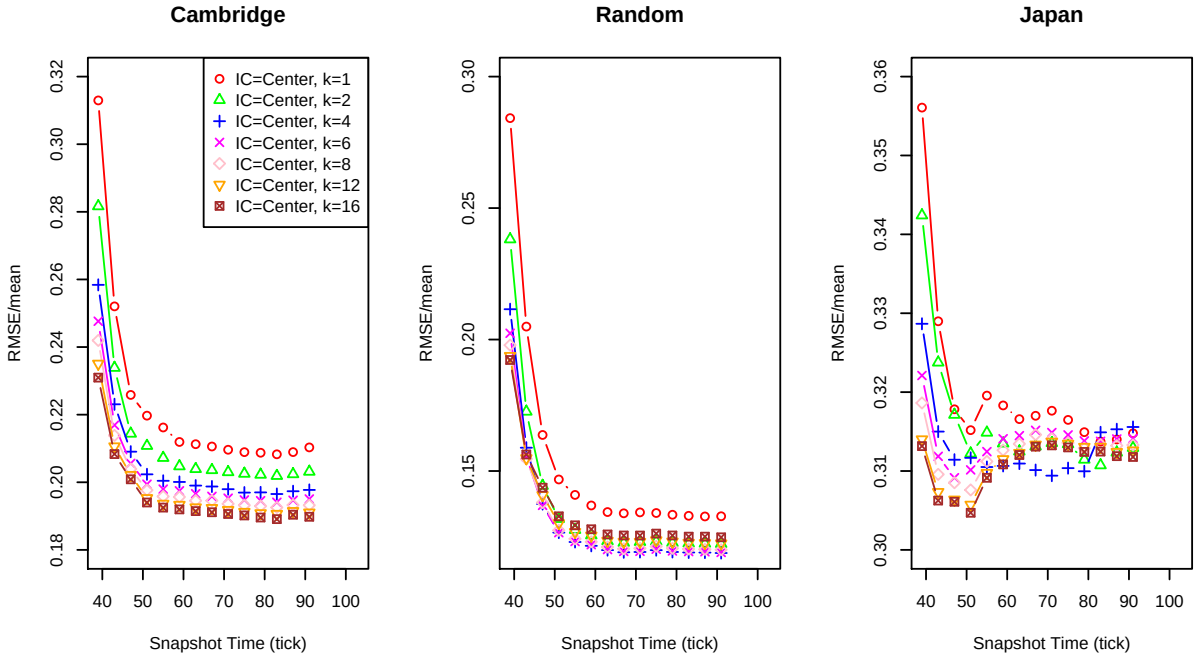



Figure 4.16. ST a$k$-Shell RMSE using $r = 48$ and 128K sensors and an IC located at the center of the window

### 4.3.5 RMSE Comparison of ST Shell and ST a$k$-Shell

The NRMSE of ST Shell compared to ST a$k$-Shell is investigated. In Figure 4.17, which shows both ST Shell and ST a$k$-Shell for an IC at the center of the window, all the tested configurations of ST a$k$-Shell have lower NRMSE values than ST Shell for the Cambridge and Random data sets and for Japan both configurations of ST Shell had higher NRMSE than ST a$k$-Shell while the phenomenon was changing and then ST Shell performed slightly better than ST a$k$-Shell for configurations using the same radius. When using an IC at the end of the window, shown in Figure 4.18, ST Shell performed better initially than ST a$k$-Shell in response to the changing phenomenon, and later after the phenomenon change slowed ST a$k$-Shell had a lower RMSE than ST Shell.

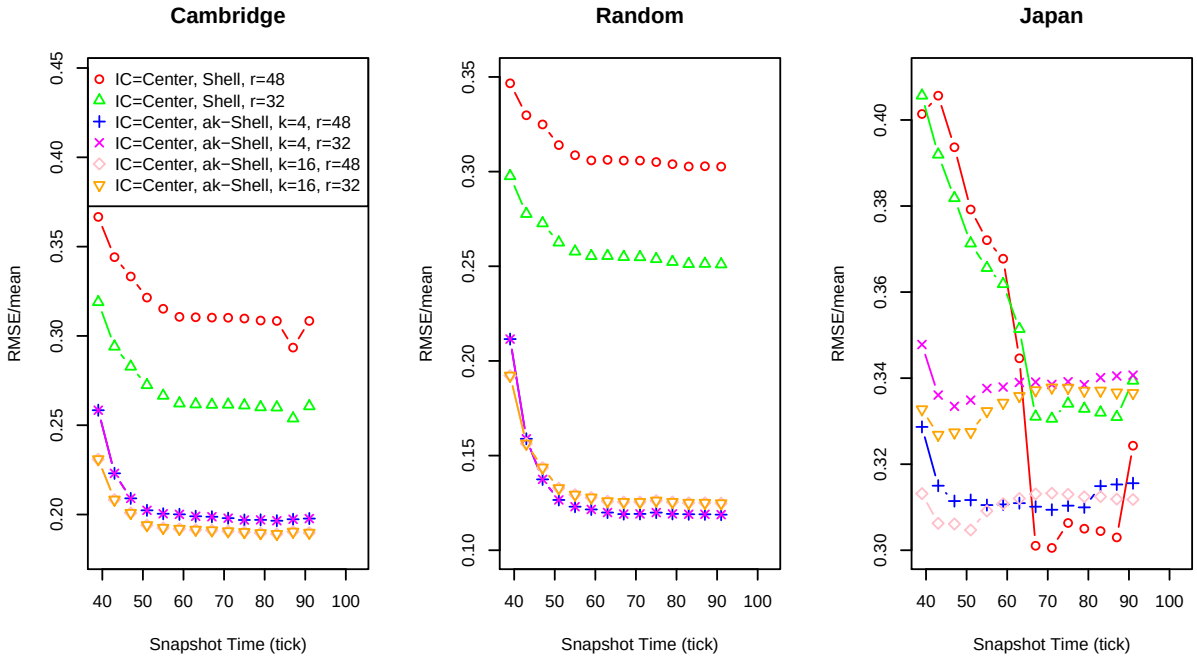



Figure 4.17. ST a$k$-Shell and ST Shell RMSE using, $a = 4$, $p = 4$, 128K sensors, and an IC located at the center of the window

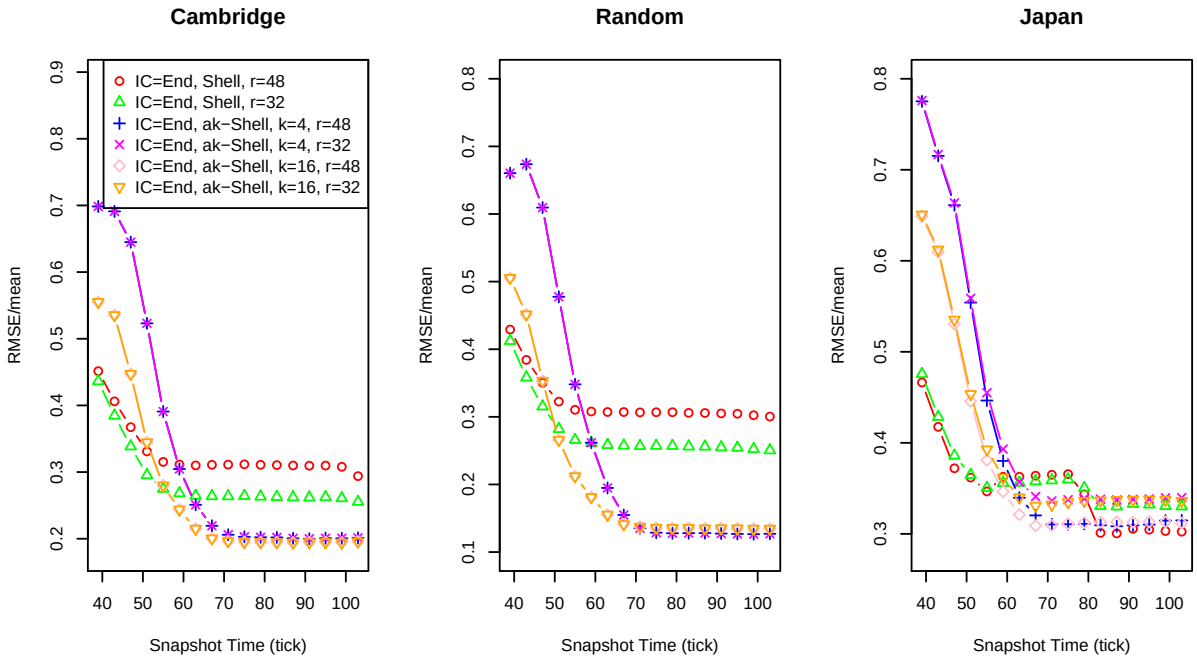



Figure 4.18. ST a$k$-Shell and ST Shell RMSE using, $a = 4$, $p = 4$, 128K sensors, and an IC located at the end of the window

### 4.3.6 IC location: Center vs. End of Window

We investigate the impact of the chosen IC, either the center or end of the window, on RMSE and compare the results in Figure 4.19. The results are presented for ST a$k$-Shell using $r = 48$ and $k \in \{4, 16\}$ and show that using an IC at the center of the window yields a lower RMSE than an IC at the end of the window. The difference in RMSE between IC end and center is largest while the phenomenon is changing rapidly and the RMSE of IC end approaches the performance of IC center when the phenomenon is static. A lower RMSE when the IC is located at the center vs. end is expected behavior for IDW as it is an *interpolation* method based on weighted averaging.

While an IC at the end of the window aims to predict the most current state of the phenomenon, this comes at an increase in RMSE compared to summarizing the window in the center since less data 'close' to the IC is available, and thus, the results are more skewed towards older data. The IC can be specified by the user to an integer time point within the

window, thus, depending on the phenomenon, its current behavior, and the importance of RMSE to the user, the user could choose the desired location for their application.

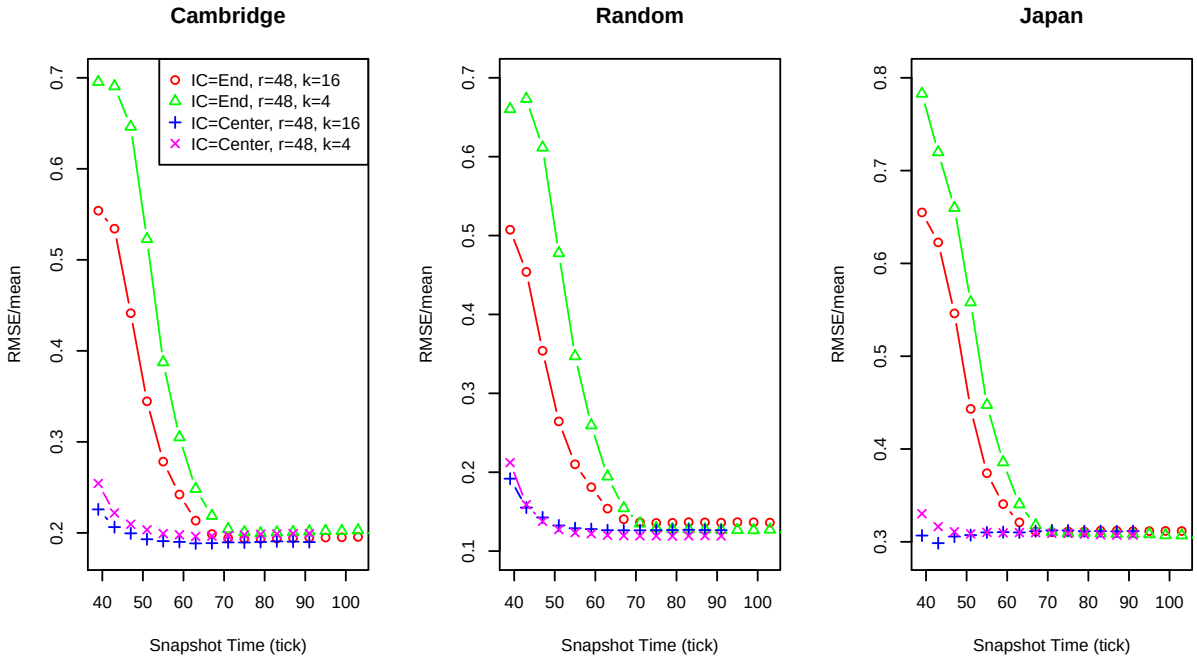



Figure 4.19. ST a$k$-Shell RMSE comparing ICs

### 4.3.7 Adjusting Phenomenon Rate of Change

In order to investigate the impact of the rate of phenomenon change and the ST anisotropy ratio, the the phenomenon was simulated at varying rate of change. A faster evolving phenomenon exhibits greater change over less time. To speed up the evolution of the (real) phenomenon, new streams were generated that captured greater change over a shorter period of time. This was accomplished by only using observations from $1/2$ and $1/3$ of the ticks and scaling the timestamps by dividing each by 2 and 3, respectively. This results in streams that simulate the phenomenon evolving two to three times faster than the original velocity. We chose an IC at the end of the window since the phenomenon develops faster and therefore, we want to represent the latest point in time of a window. The results for using ST a$k$-Shell with $r = 32$, $k = 4$, 128K sensors, and a drop rate of $1/2$

are presented in Figure 4.20 with timestamps mapped to the original snapshot ticks. The results confirm that as the rate of phenomenon change increases, $a$ should be decreased to minimize RMSE.

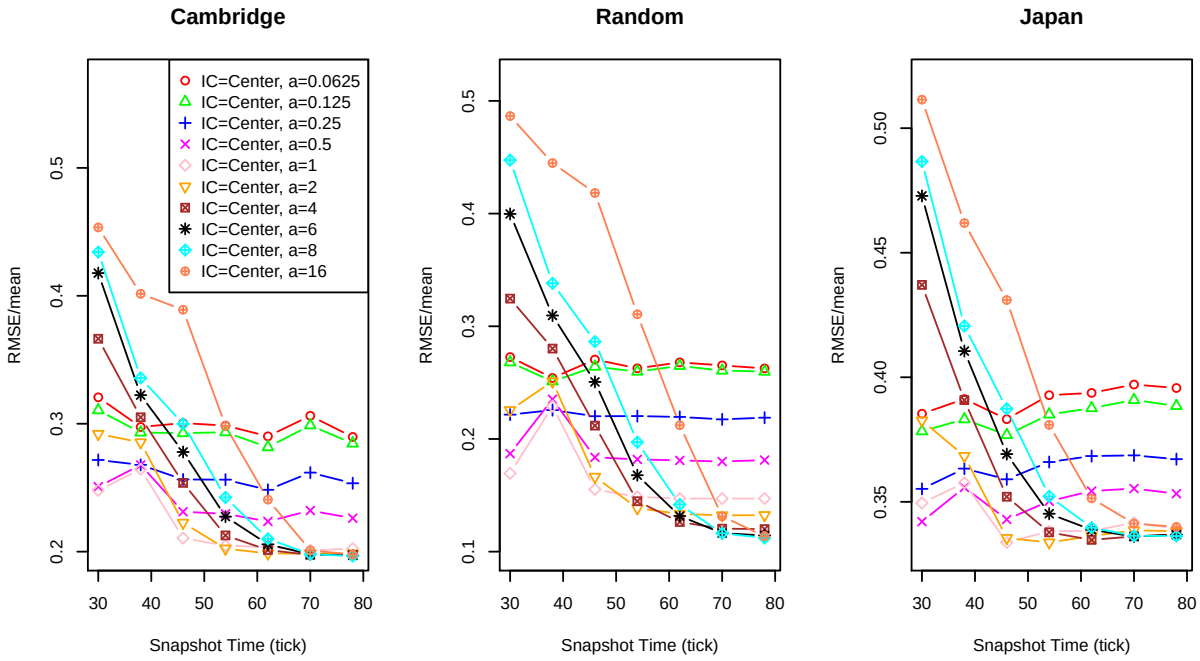



Figure 4.20. RMSE results from increasing phenomenon rate of change by dividing timestamps by 2 using ST a$k$-Shell with $r = 32$, $k = 4$ plot, IC at the center of the window, and 128K sensors

### 4.3.8 Adjusting Window Size

We investigated the impact of the window size on RMSE. In order to test the impact of window size rather than the number of tuples in a window, a constant stream tuple density of 512 tuples per tick was used. Having a fixed tuple density allows the impact of window size over a given stream to be investigated. Window sizes $w \in \{1, 2, 4, 8, 16, 32\}$ were tested, yielding $n \in \{512, 1024, 2048, 4096, 16384\}$ tuples per window. An IC at the end of each window was used and results are shown in Figure 4.21. The results show that while using a large window size of $w = 32$ yielded the worst results in time of change, it performed best when the phenomenon was static.

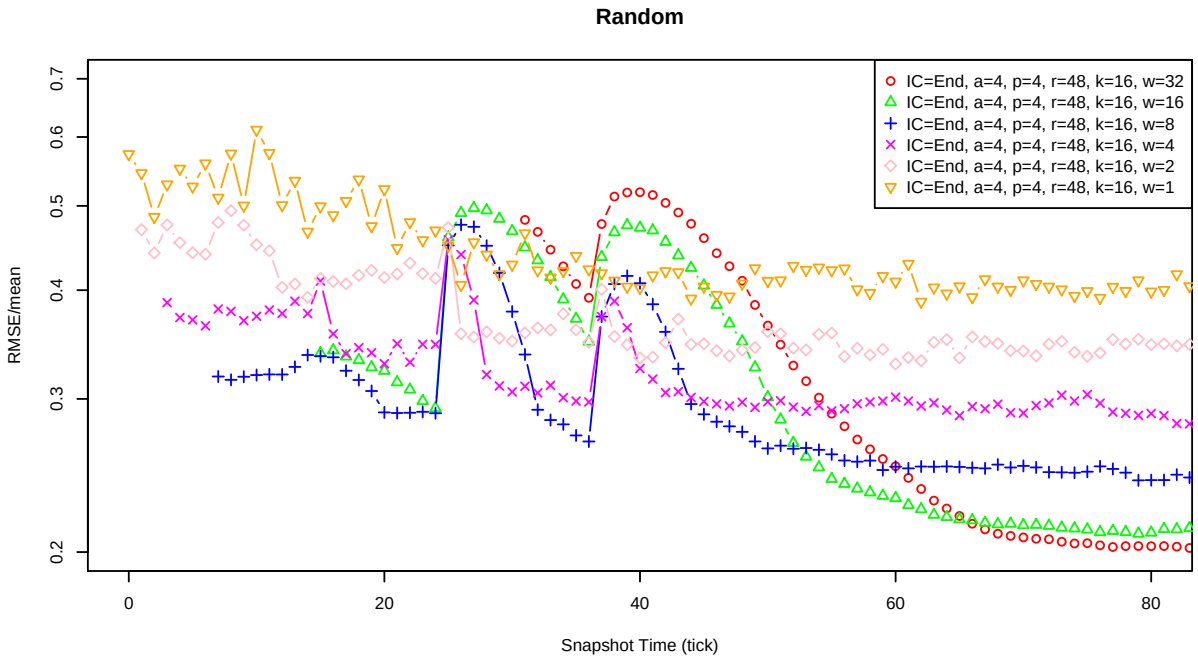



Figure 4.21. Adjusting window size with constant tuple density of 512 tuples/tick

### 4.3.9 RMSE Summary

In the RMSE tests, the RMSE of ST Shell and ST a$k$-Shell was investigated for different parameter configurations. The configuration producing the lowest RMSE comparing different ST anisotropy ratio ($a$) and ST-IDW power ($p$) value was found to be $a = 4$ and $p = 4$. Using ST Shell, $r = 16$ produced the lowest RMSE for the Cambridge and Random data sets and $r = 48$ performed best for the Japan street network data set. For ST a$k$-Shell, $r = 32$ and $r = 48$ performed similarly for the Cambridge and Random data sets and for Japan $r = 48$ performed resulted in a lower RMSE than $r = 32$. For the parameter $k$, $k = 16$ had the lowest RMSE in response to change in the phenomenon and had similar RMSE compared to those for other values of $k$ when the phenomenon was static. The difference in RMSE between IC end and center is largest while the phenomenon is changing rapidly, with an IC located at the center of the window having lower RMSE than an IC at the end of the window. For an IC at the center of the window, all the tested configurations of ST a$k$-Shell have lower RMSE than ST Shell for the Cambridge and Random data sets

and for Japan both configurations of ST Shell had higher NRMSE than ST a$k$-Shell while the phenomenon was changing. When using an IC at the end of the window, after the phenomenon change slowed ST a$k$-Shell had a lower RMSE than ST Shell.

As the Cambridge, Japan, and Random data sets are representative of different sensor distributions, insights from the RMSE tests are relevant to other sensor deployments. The tests demonstrate the suitability of using ST a$k$-Shell with a neighborhood defined by $r$ and $k$ for both dense and sparse distributions of observations. For a sparse distribution, $r$ should be sufficiently large such that all cells in the area of interest can find sample points. If data are dense, $k$ can be reduced from 16. To minimize RMSE an IC at the center of the window should be used, while if representing the latest predicted state of the phenomenon, an IC closer to the end of the window should be used. From these tests, we determined the configurations that produced the lowest RMSE to evaluate further with respect to runtime.

## 4.4 Runtime Investigation

After we determined appropriate parameters that can produce query results with an appropriate RMSE we now evaluate the runtime and throughput performance of ST Shell and ST a$k$-Shell.

### 4.4.1 Impact of Radius Size $r$ for ST Shell on Runtime

First, we investigated the impact of the neighborhood radius, $r$, on runtime for ST Shell using $r \in \{16, 32, 48\}$. The results for ST Shell using 128K sensors and an IC at the center of the window are plotted in Figure 4.22, which shows consistent runtime performance for ST Shell across all street networks and snapshot times. As expected, increasing the value of $r$ results in an increase in runtime. In Figure 4.23, the scalability of ST Shell is investigated as the number of sensors in a window increases, and the constant slope for each $r$ confirms linear scalability of ST Shell with respect to the number of sensors.

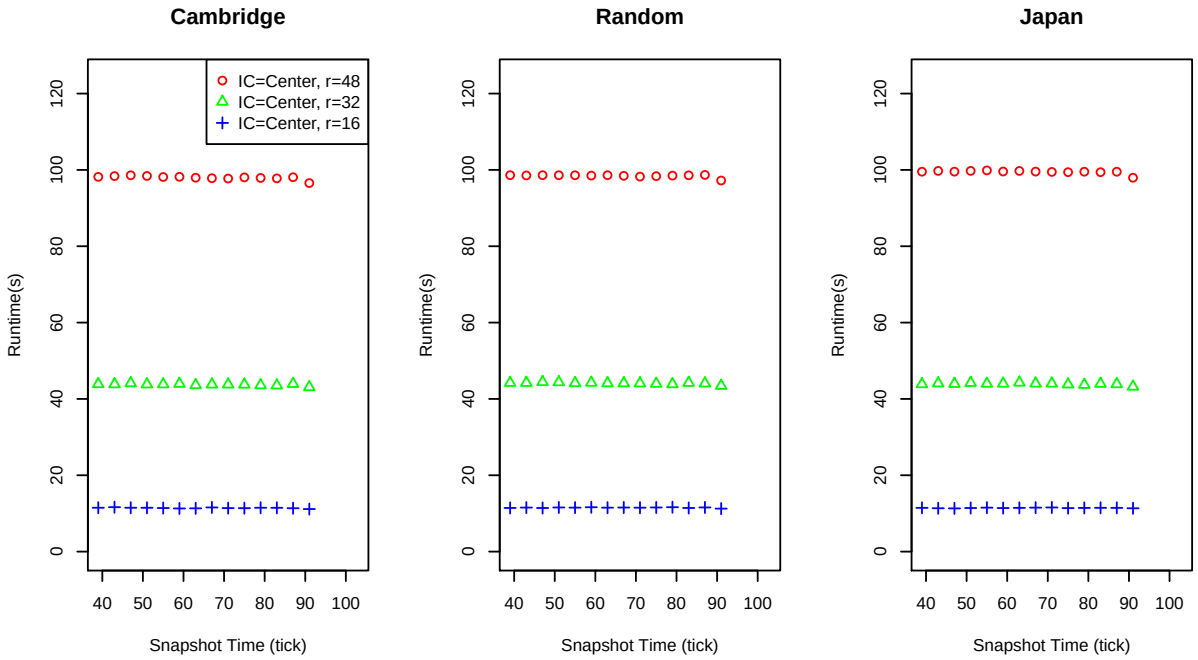



Figure 4.22. ST Shell runtime using 128K sensors, testing $r \in 16, 32, 48$ and an IC located at the center of the window

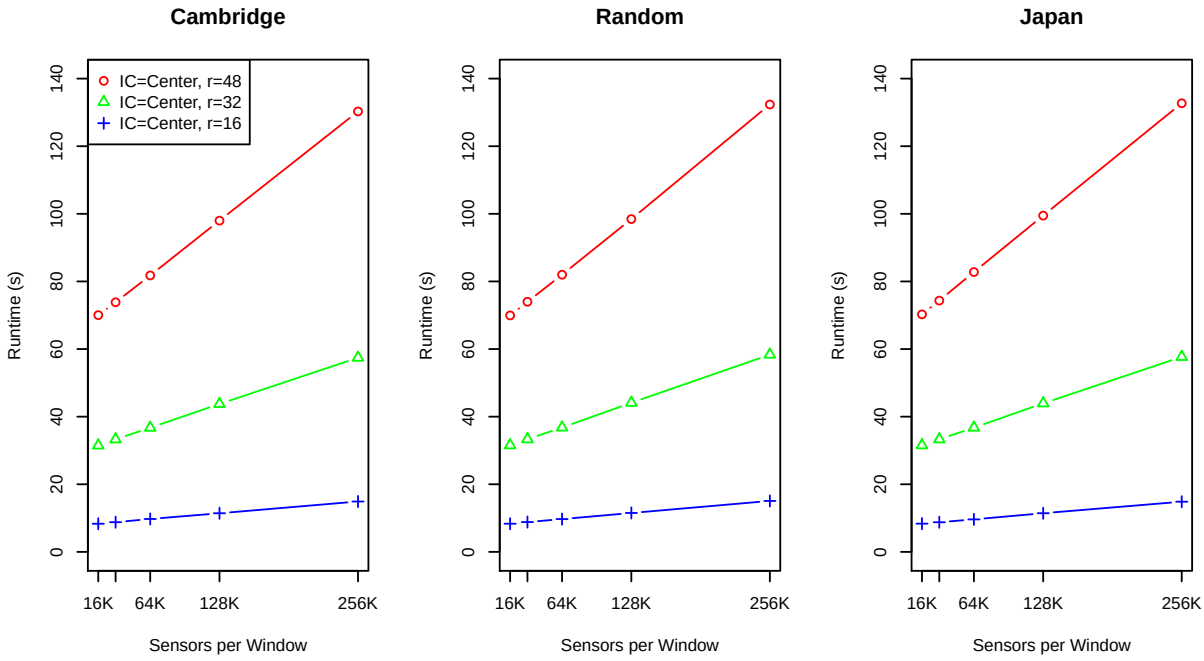



Figure 4.23. ST Shell runtime vs. number of sensors, testing $r \in 16, 32, 48$ using an IC located at the center of the window

### 4.4.2 Impact of Parameter $r$ and $k$ for ST a$k$-Shell

In ST a$k$-Shell, the parameter $k$ is used to limit the search for tuples and terminate the expanding ST search once $k$ tuples are found, whereas $r$ is used to define the radius of the expanding search sphere and bounding ST cylinder. The combined runtime plots for $r \in \{16, 32, 48\}$ and $k \in \{4, 6, 8\}$ for configurations $r \times k$ are shown with respect to snapshot time (Figure 4.24) and the number of sensors in a window Figure 4.25.

The runtime of ST a$k$-Shell depends significantly on the spatial distribution and number of sensors. This is most evident in Figure 4.24 where runtime for the Random data set is impacted most significantly by $k$ whereas for the Japan data set runtime is impacted most by $r$. The Random data set (minimal spatial skew) and Japan data set(high spatial skew) result in $k$ being the dominating termination condition for the Random data set and $r$ for the Japan data set.

The Cambridge data set results show that the runtime for this distribution of sensor is impacted by both $k$ and $r$, but most significantly by $k$. This is due to the distribution of observations on the Cambridge street network, which is similar to the random distribution over land portion but there are still large water bodies without any observations. Thus, in areas near the street network search terminates after $k$ points and in areas over water interpolation of some cells requires expanding to larger radius to find $k$ tuples. Since runtime plots for Cambridge are not as closely grouped by $k$ as in Random, this means that for $r \in \{16, 32\}$ in Cambridge there are some cells where $k$ tuples are not within $r$ and search terminates by $r$, but a much smaller area than for Japan.

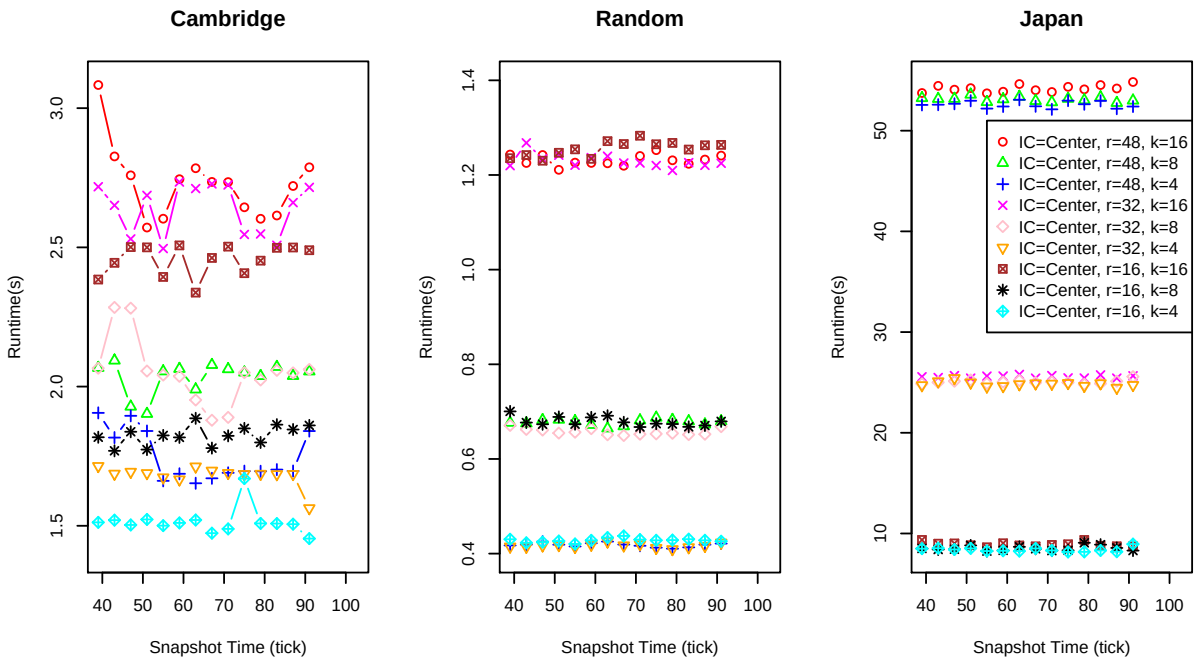



Figure 4.24. ST a$k$-Shell runtime using $r = 32$ and 128K sensors, testing $r \times k$ for $r \in \{16, 32, 48\}$ and $k \in \{4, 8, 16\}$ and an IC located at the center of the window

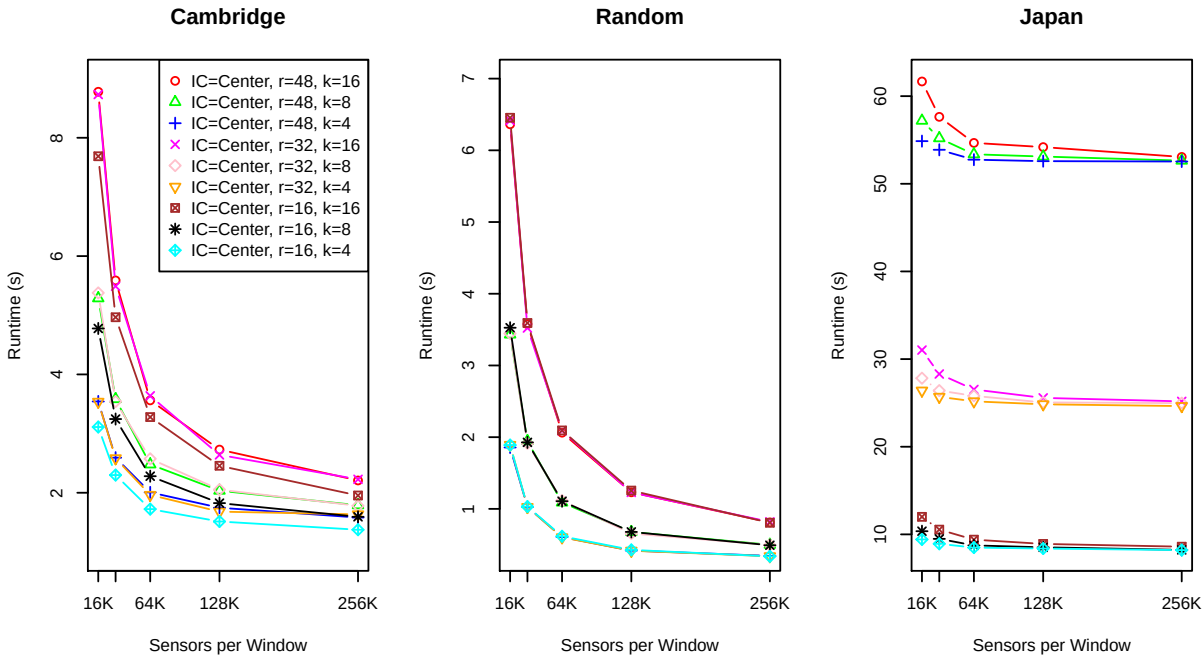



Figure 4.25. ST a$k$-Shell runtime vs. number of sensors, testing $r \times k$ for $r \in \{16, 32, 48\}$ and $k \in \{4, 6, 8\}$ and IC located at the center of the window and averaged over all snapshots

### 4.4.3 Snapshots: Center vs. End of Window

We investigate the impact of the chosen IC, either the center or end of the window, on runtime and compare the results in Figure 4.26. The results are shown for ST a$k$-Shell using $r = 48$ and $k \in \{4, 16\}$. The results show that for the configurations tested, runtime performance is consistent between an IC at the End and Center of the window. Note, that in [165] where $a = 1$, $p = 2$, and Cartesian distance were used, having an IC at the end of the window was found to be faster than at the center, but now using $a = 4$, where search is performed at four times the rate in time as in space, and there is increased computational time spent on distance calculation using haversine and $p$, the difference in runtime performance for and IC at the Center and End of the window is no longer noticeable.

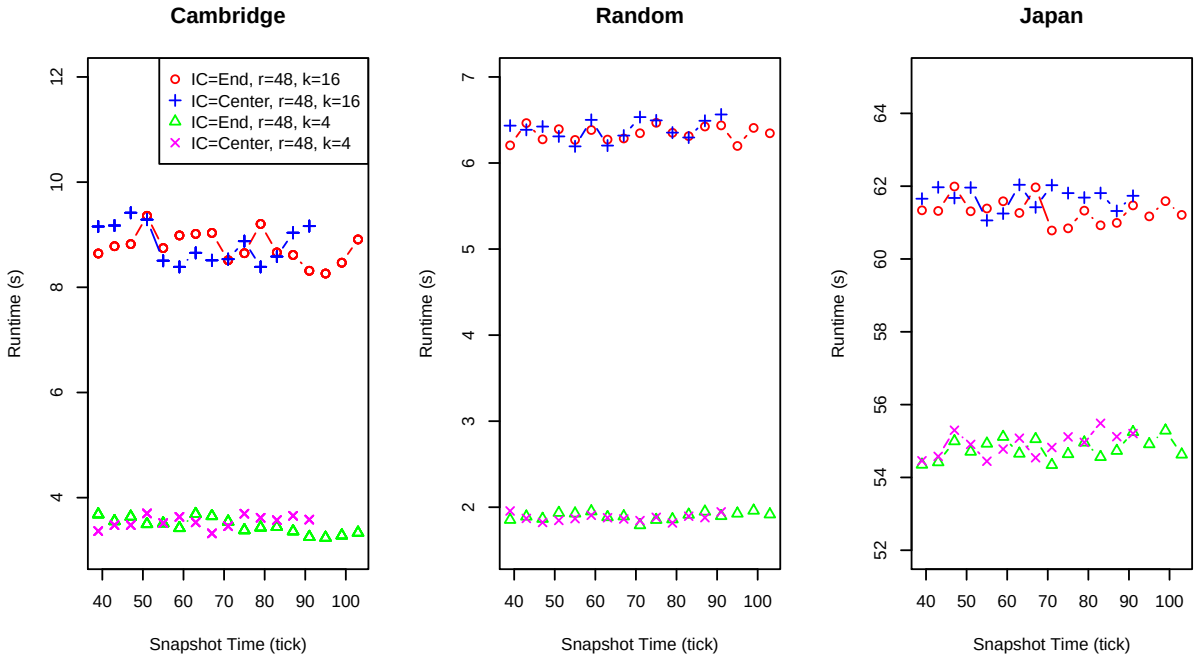



Figure 4.26. ST a$k$-Shell runtime comparing ICs for 16K sensors

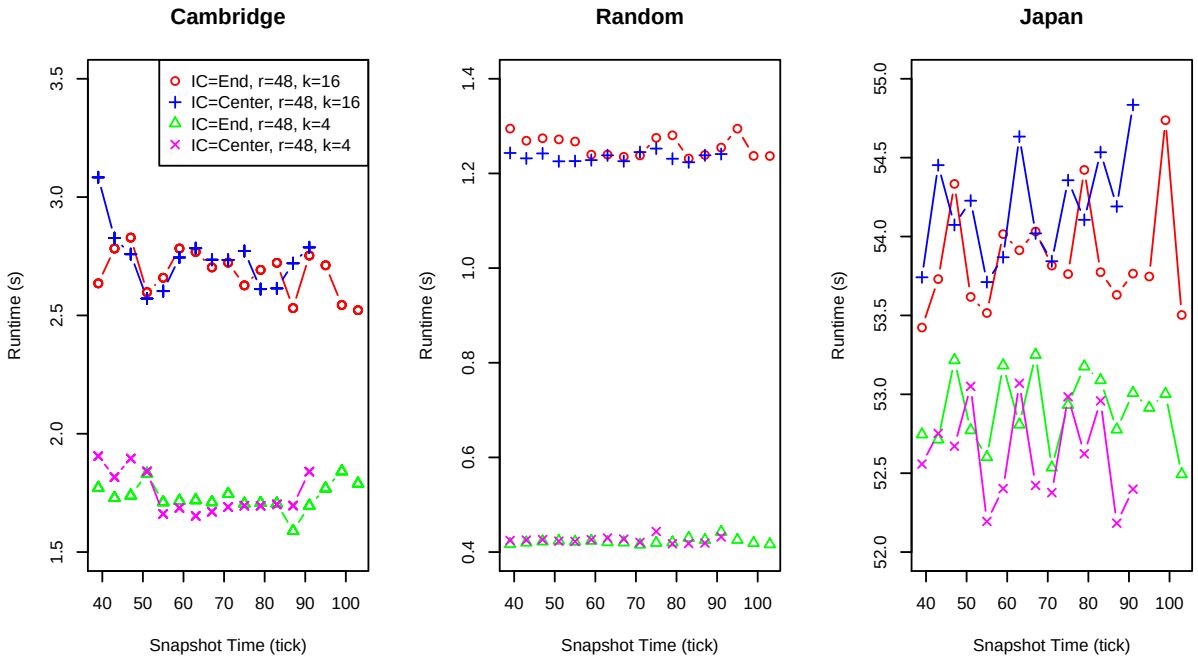



Figure 4.27. ST a$k$-Shell runtime comparing ICs for 128K sensors

## 4.5 Summary

### 4.5.1 RMSE vs. Runtime

We evaluated the trade-off between runtime and accuracy for ST a$k$-Shell for $r \in \{16, 32, 48\}$ and $k \in \{4, 8, 16\}$ for $r \times k$. Results for anisotropy ratio $a = 4$ and power $p = 4$ are presented for an IC at the center (see Figure 4.28 and Figure 4.29) and end of window (see Figure 4.30 and Figure 4.31) for a snapshot at tick 51 and tick 83. In terms of runtime for ST a$k$-Shell, as shown in all RMSE vs. runtime plots (Figures 4.28 to 4.31), an increase in the number of sensors observations per window reduces the runtime since the search for each cell is shorter as data are more dense and the number of cells to interpolate is constant.

ST a$k$-Shell with an IC at the center of a window performs consistently well with regard to runtime and RMSE as the number of observations per window increases, both when the phenomenon is changing quickly in Figure 4.28 and is more stationary as in Figure 4.29. Larger data sets provide both low runtime and low RMSE since $k$ samples can be found in

closer proximity to each cell when data are more dense, which reduces search effort and RMSE as samples are closer to prediction locations.

ST interpolation is more challenging when using an IC at the end of the window as the samples in a window only capture the state of the phenomenon prior to the IC. When using an IC at the center of the window, there are samples both before and after the IC, which enables ST interpolation to predict values between observed data. The results for an IC at the end of the window (see Figure 4.30) indicate that when the phenomenon is changing quickly, using an IC at the end of the window has a higher RMSE compared to an IC at the center of the window (Figure 4.28). Additionally, the results for an IC at the end of the window show a change in concavity of the graphs between snapshots for tick 51 (Figure 4.30) and tick 83 (Figure 4.31). This trend for an IC at the end of the window is most clearly seen in the Random data set where at tick 51 (Figure 4.30), an increase in data points above 64K resulted in an increase in NRMSE, while at tick 83 (Figure 4.31), an increase in data points resulted in a decrease in NRMSE.

The RMSE vs. runtime plots identify the trade-off between runtime and accuracy based on different parameter configurations of ST a$k$-Shell and the number of sensor observations per windows. Depending on a user's prioritization of low runtime and high accuracy, the parameters of $r$ and $k$ can be chosen for the distribution of sensors in the data set.

- **Cambridge data set**

  For the Cambridge data set, the configuration of $r = 32$ and $k = 16$ produced the lowest NRMSE with a runtime of 2-3 seconds for 128K sensors. In order to reduce runtime further, $r$ or $k$ can be decreased; a reduction in $k$ offers the most significant reduction in runtime at the cost of significantly increasing NRMSE.

- **Random data set**

  For the Random data set, using $r = 16$ and $k = 16$ provided the most accurate results and the runtime was 1-3 seconds. Reducing $k$ from $k = 16$ increased runtime at the

cost of an increase in NRMSE while increasing $r$ did not have a noticeable impact on runtime.

- **Japan data set**

  In the Japan data set, using $r = 48$ and $k = 16$ yielded the lowest NRMSE and had a runtime around 60 seconds due the sparse sampling based on rural street networks and therefore large radius per estimated value. Decreasing $r$ improved runtime since the search for sample points was terminated after expanding to a smaller radius ($r = 16$ or $r = 32$ compared to $r = 48$) in areas where samples were sparse, including areas over the ocean and remote mountains. The reduced runtimes, at 20-30 seconds for $r = 32$ and around 10 seconds for $r = 16$, came at the expense of increased NRMSE, as for areas that had no samples within $r$, an interpolated value for the cell could not be calculated based on ST neighboring samples. Instead, the value of these missing cells was replaced by the mean of the cells that were interpolated. While reducing $r$ lead to a reduction in overall NRMSE, the RMSE of cells having $k$ samples within $r$ is not impacted.

Overall, the recommended configurations that produced the lowest RMSE are: $r = 32$ and $k = 16$ for Cambridge, $r = 16$ and $k = 16$ for Random, and $r = 48$ and $k = 16$ for Japan. Runtime for these configurations is around 2-3 seconds for Cambridge, 1-3 seconds for Random, and around 60 seconds for Japan. The NRMSE is lowest in Figure 4.29 with around 0.185 for Cambridge, 0.1 for Random, and 0.3 for Japan.

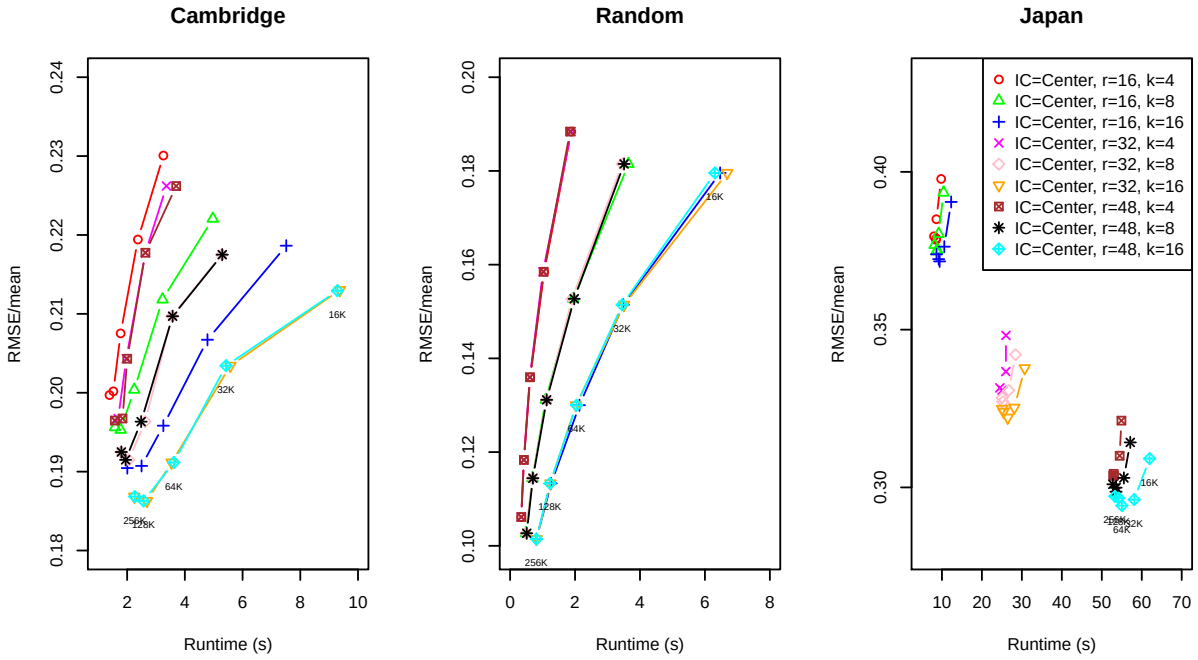



Figure 4.28. ST a$k$-Shell runtime vs RMSE using an IC located at the center of the window for a snapshot at time t=51

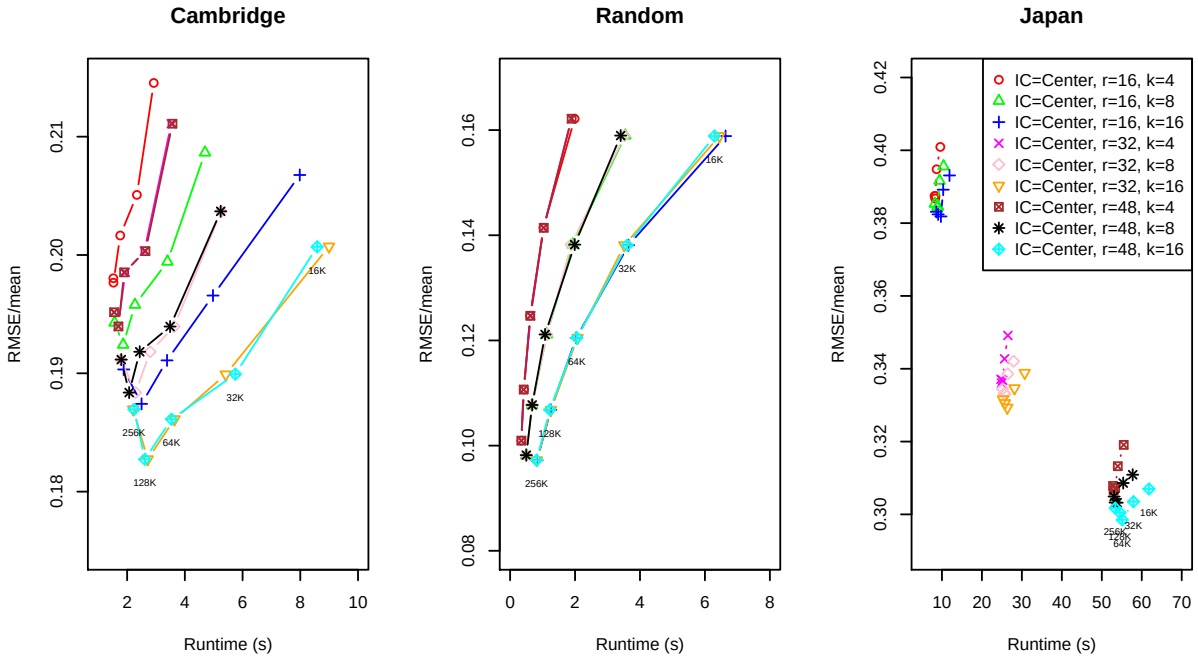



Figure 4.29. ST a$k$-Shell runtime vs RMSE using an IC located at the center of the window for a snapshot at time t=83

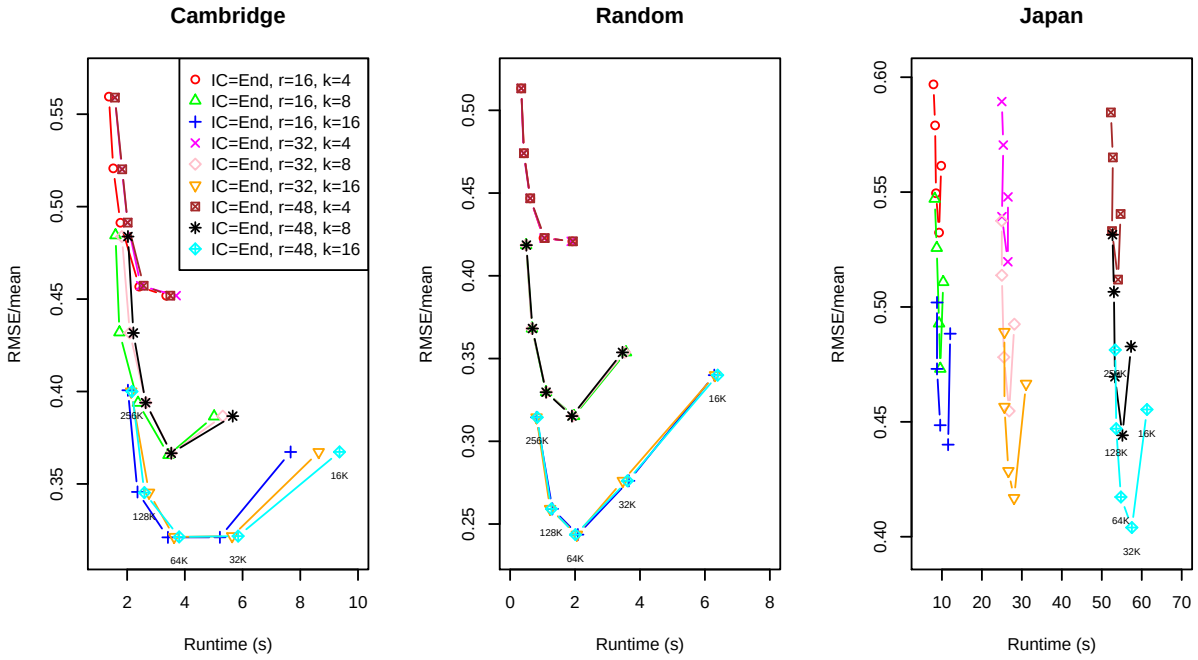



Figure 4.30. ST a$k$-Shell runtime vs RMSE using an IC located at the end of the window for a snapshot at time t=51

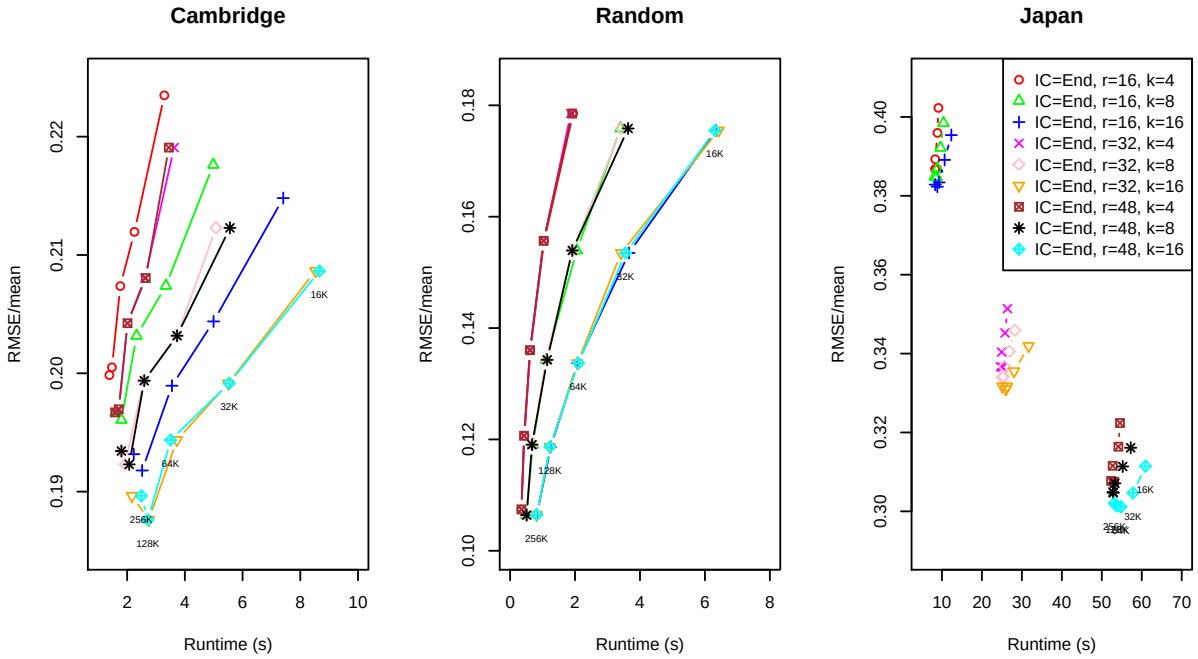



Figure 4.31. ST a$k$-Shell runtime vs RMSE using an IC located at the end of the window for a snapshot at time t=83

### 4.5.2 Performance Comparison to Related Stream-Based ST Interpolation

To evaluate the runtime performance of our approach using stream-based ST-IDW, we compared our runtime results to the stream-based ST Kriging approach used by Lorkowski and Brinkhoff [95, 96]. Performance tests [96] demonstrated roughly linear runtime scalability with the number of observations $n$ and with the number of prediction locations $m$, meaning that both ST-IDW and ST stream-based Kriging have comparable computational complexity of $O(mn)$. The results [95, 96] showed that for a set of 400 sample points, partitioned into batches of 10 points, cumulative computational time takes over 30 seconds for the combined result for a $150 \times 150$ raster grid. Scaling these results up to a larger grid of $500 \times 500$ pixels and 128K tuples yield an expected runtime of over 30 hours whereas our approach using ST a$k$-Shell requires around 2 seconds, over four orders of magnitude faster.

$$\frac{500 \times 500 \text{ pixels}}{150 \times 150 \text{ pixels}} \times \frac{131,072 \text{ tuples}}{400 \text{ tuples}} \times \frac{30 \text{ s}}{3,600 \text{ s/h}} = 30.34 \text{ h}$$

### 4.5.3 Discussion of Results

For the RMSE tests, $a = 4$ and $p = 4$ were found to be the best tested parameters for ST anisotropy and ST-IDW power, respectively. Using an IC at the center of the window results in the lowest RMSE. The ST a$k$-Shell approach performs better than ST Shell for an IC in the center of the window, and ST a$k$-Shell was only slightly worse than ST Shell during the expansion phase of the phenomenon. Increasing the change rate of the phenomenon showed that $a$ should be decreased as the rate of change of the the phenomenon increases. The optimal value of $r$ and $k$ depends on the spatial arrangement of sensors, with small values of $r$ being best for dense distributions (Cambridge and Random) and larger values of $r$ being best for sparse distributions(Japan), and large values of $k$ result in a decrease in RMSE for all distributions. Larger values of $k$ were tested and did not show significant improvement, or were even worse.

In the runtime tests, the termination condition that has the highest impact on runtime for ST a$k$-Shell depends on the distribution of the sample points. When points are more uniformly distributed, the search is terminated once $k$ sample points are found rather than exhaustively searching all cells in $r$. When points are spatially skewed, a larger $r$ is needed to find $k$ sample points, and even that value of $r$ may be insufficient to interpolate all portions of the result. For the more spatially uniform Random and Cambridge data sets, the runtime results were clustered by $k$, meaning the search was most often terminated by $k$ and for the spatially skewed data set of Japan, the results were clustered by $r$, indicating that in many areas $k$ points were not found within $r$. In contrast to ST Shell where an increase in sample points resulted in an increase in runtime, for ST a$k$-Shell an increase in observations per window results in a decrease in runtime.

Depending on the desirability of minimizing RMSE or runtime, specific values of $r$ and $k$ can be chosen for each data set. In general, the more sensors the better as an increase in sensors resulting in both a decrease in runtime and RMSE.

Our performance tests show that snapshot queries using ST a$k$-Shell run in under 2.5 seconds per window; the configuration for this result is that the ST search radius is 48 cells around a cell to be interpolated, a query window consists of 32 time units, roughly 256K observations are available, using the Cambridge and Random sensor distribution data sets.

In comparison to related work [95, 96] on stream-based ST interpolation, our work is expected to be over four orders of magnitude faster for a similar number of sensor observations and size of prediction grid.

## 4.6 Conclusion

This chapter examined the use of the STI-SQO framework to process massive asynchronous observation streams from mobile sensors using ST interpolation over sliding windows to represent continuous phenomena. Performance evaluation was conducted using the Fukushima nuclear event in March 2011 as an example of a dynamic phenomenon for

the test data set. It was found that ST-IDW can be used to generate $500 \times 500$ pixel spatio-temporal snapshots in less than 2.5 seconds with 256K tuples per window. This finding confirms the hypothesis as the 100,000 tuples/s throughput target is achieved and the near real-time target of representations generated in 1-3 seconds.

The challenges of asynchronous sensing, mobile sensing, capturing the ST continuity of the phenomenon, and high data throughput are addressed by the contributions of the approach. Asynchronous sensing is addressed using windows to group tuples and ST-IDW with an ST anisotropy ratio to weight observations by location in space and time. The challenge of mobile sensors is addressed by using ST-IDW which has $O(n)$ time complexity, even when observation locations change, and an ST grid index that keeps track of observation locations and disregards sensor identity. The ST continuity of a phenomenon over a window is captured using a single snapshot at a user-specified Interpolation Center (IC) or a movie query consisting of multiple snapshots. High data throughput is achieved by using a DSE with an in-memory index and the ST a$k$-Shell algorithm which performs faster as the amount of data in a window increases. We address the resource consuming identification of nearby tuples in both space and time during ST-IDW execution with the contribution of the ST grid index and two shell-based search templates: the cylindrical, ST Cylindrical Shell Template (CST) and the concentric, ST Nested Shell Template (NST). The NST allows quickly determining the nearby tuples using ST, concentric search and is employed by ST a$k$-Shell to search the ST cuboid of the ST grid index. The ST grid index consists of time subblocks as the smallest temporal partition of the ST grid index and make it possible to achieve three tasks: a.) searching proportionally in both space and in time using Isotropic Time Cells (ITCs) with the NST used by ST a$k$-Shell., b.) discarding outdated tuples with each window slide, and c.) reusing the ST grid index between consecutive windows in sliding window queries.

The STI-SQO framework has demonstrated it's ability to perform near real-time ST interpolation of massive numbers of sensor observation streams.

# CHAPTER 5
# A STREAM QUERY OPERATOR FRAMEWORK FOR EVALUATING PREDICATES OVER CONTINUOUS SPATIO-TEMPORAL FIELDS

## 5.1 Motivation

Consider the Fukushima Daiichi nuclear disaster scenario described in Chapter 3, where the distribution of radiation is a continuously evolving ST phenomenon. While being able to monitor near real-time representation of the entire continuous phenomenon is useful in many applications, users are often interested in only being alerted when hazardous threshold conditions are breached.

Therefore, analysis operators that continuously re-evaluate over such a phenomenon are desirable. An analysis operator can be a simple threshold operator in which a user specifies a condition, i.e., whether the user is interested in values of the phenomenon that are above, below, or equal to this threshold, and the values satisfying the condition are selected for the user. For example, consider Query 5.1 which retrieves the caesium-137 values satisfying the condition of being greater than $1\,\mathrm{Bq/m^3}$ (Becquerel/$\mathrm{m^3}$) at each point within the boundary of Japan. The result should be retrieved as a stream of new rasters called `hazardous_radiation_deposition`.

Mathematically defined, a threshold operator is a logical predicate [37]. A predicate is a function that evaluates a value to either true or false depending on whether the value makes the condition true or false. For instance, a predicate over a field representation of ST phenomenon is a unary field operator and the Boolean proposition is applied to each element of a field's domain, $r = P_{prop}(f)$.

Query 5.1. Example predicate query

```
SET @Grid = -- Points in the raster grid --
SET @Japan = -- Polygon of Japan --
SET @IC = 5 min
SET @Condition = x > 1

SELECT
        RASTER(@Grid,
    PREDICATE(
        ST-IDW(@Grid, sensors.loc, sensors.val, @IC),
        @Condition))
        AS hazardous_radiation_deposition
FROM
        sensors WINDOW 5min SLIDE 1min
WHERE
        sensor_type = radiation
    AND INSIDE(sensors.loc, @Japan);
```

## 5.2 Problem Statement

The main problem with predicate stream queries over ST phenomenon is that users are interested in expressing such a queries in terms of a predicate over an ST phenomenon, but only streams of discrete sensor observations are available. This also implies that the user is more interested in retrieving entire query result *regions* of the ST phenomenon instead of filtered candidate tuples that fulfill the condition of the raw streams.

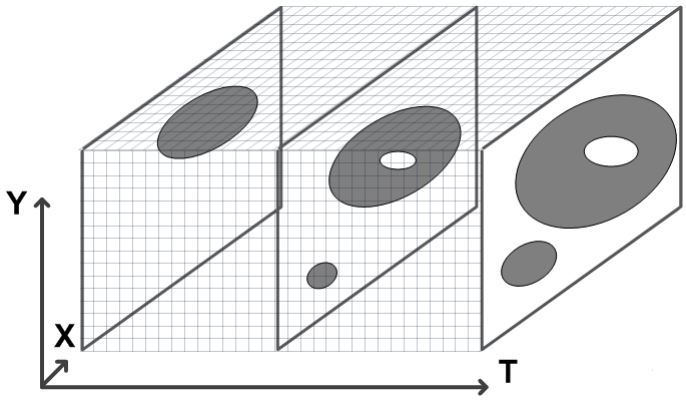



Figure 5.1. Evolving predicates result regions over time

Assuming that an appropriate data model language and query language are available to express predicates over ST fields as in [90], the problem remains that the stream query can only be *executed* over the individual, discrete sensor streams.

Continuous phenomena have several dimensions: time, space and values. Thus, predicates can be expressed over the individual dimensions or combination of the dimensions. In this chapter 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."

### 5.2.1 Challenges

- **Limiting the ST Interpolation to Predicate Result Region(s)**

  The runtime for IDW increases linearly with the number of cells interpolated, and reducing the number of cells interpolated also reduces the computational time. Existing interpolation algorithms [85, 80, 149] only focus on interpolating entire raster cells as these approaches are unable to limit interpolation to cells that satisfy a predicate, since they are not designed as predicate evaluation algorithms.

- **Accurately Identifying Predicate Regions**

  A method is needed to accurately extract predicate result regions. An approach that interpolates an entire raster first using all sample data and extracts the raster cells that satisfy the predicate produces **accurate results** in our context. In this context, we define the baseline as an approach that interpolates an entire raster first using all available sample data and extracts the raster cells that satisfy the predicate. This approach produces the most **accurate results** in our context (i.e., the best that can be achieved given a stream of samples), however, also at the potentially highest computational cost. However, any approach that targets results regions directly and avoids interpolating unnecessary cells should produce result regions that are similarly accurate.

- **Continuous Evolution of Phenomena**

  Phenomena continuously evolve over time and space, and so does the result of a value predicate as shown in Figure 5.1. When evaluating the predicate result, changes in values and topology need to be captured in the raster representation. Changes to the phenomenon require interpolation to detect that the phenomenon has changed, and those areas need to be interpolated.

- **Coordinating Parallel Phenomenon Interpolation Operators**

  Predicate evaluation requires coordinating multiple ST interpolation operators so that duplication of effort can be eliminated (e.g., two operators actually do evaluate the same result region). Additionally, if any search or traversal strategy is used, the status of each grid cell must be accessible by all operators to prevent unnecessary interpolation of grid cells.

### 5.2.2 Naïve Algorithm Establishing the Baseline

The *Naïve* algorithm establishes a baseline for result quality ad run-time. It represents a straightforward method of evaluating a value predicate over a ST continuous phenomenon. It assumes that first a continuous representation of the phenomenon is generated by interpolating a raster for the query window, and that then the value predicate is evaluated over all raster cells. The choice of an 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 is a reasonably efficient strategy to evaluate the predicate. However, if only a small percentage of the field qualifies for the predicate the Naïve algorithm has estimated many cells that are not part of the query result. It is more efficient to identify these qualifying regions and restrict the ST predicate evaluation to the result subfields.

We focus on algorithms that perform predicate evaluation over a limited area and for the remainder of this chapter and compare the studied algorithms with the Naïve baseline in the performance section found in Section 5.8.

## 5.3 Related Work

The problem of continuously evaluating predicates over continuous phenomena has only been addressed in a limited fashion in prior work.

### 5.3.1 Evaluating Spatio-Temporal Predicates over Continuous Phenomena

Currently, stream query predicate operators are available only over raw ST streams [14, 32, 39, 48, 50, 65, 71, 77, 105, 106, 110, 109, 119, 175] and there is limited support for ST continuity of phenomena provided by the Nile-PDT system [7, 8]. Nile-PDT identifies candidate phenomena from sensors having similar values and uses a polygon to connect the sensors detecting a phenomenon. In the Nile-PDT system [7, 8] a confidence function is used to choose which sensors to join with and whether sensors are part of the same or different phenomena, but there is no assessment of the accuracy of the phenomena regions, in the form of polygons, detected. For predicates over ST streams, delivering accurate query results during load shedding, such as k-Nearest Neighbors (kNN) queries for moving objects, is investigated to assess the similarity of the results [106, 119, 8]. The work in this dissertation is the first work to address efficient and accurate stream query processing of predicates over continuous phenomena in near real-time.

### 5.3.2 Spatio-Temporal Predicates over Moving Object Data Streams

Moving objects observed via near real-time location update streams are common scenarios for DSEs. 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 five nearest cars to landmark A"), or Aggregate Nearest Neighbor (ANN) queries (e.g., "for a set of cars and a set of landmarks, what is the distance to the nearest landmark to each car in the last window") [39].

Contributions with regard to predicate operator implementations for moving object streams have been made in the research DSEs Place [105, 110], SINA [108], Nile [66],

CAPE [141], and others. In moving object data streams, spatial predicates are evaluated trajectories of individual objects, that is, queries such as "which are the current kNN neighbors of the police car" are evaluated. Thus, collocated objects need to be identified quickly. These stream queries are also supported often by a shared, in-memory spatial grid index [56, 108, 105, 110, 106] since grid indices have better behavior under such high update load and the implicated insertion and deletion from the spatial index per window.

Additionally, if the trajectory of an object is of interest, information of individual objects' locations needs to be maintained for queries over time across stream query windows, which is not the case with moving sensors for continuous phenomena. Building pipelined query operators and plans that can be parallelized and evaluated incrementally has been one of the major challenges [39]. However, tracking individual objects across windows is a fundamentally different task than spatially interpolating all sample points within a window, and then evaluating a value predicate on this summary.

## 5.4 Hypothesis

Our hypothesis is that a stream query operator framework for predicate evaluation over continuous ST phenomena with cumulative areas predicate result regions between 40% and 10% of the entire representation area, using a predicate evaluation algorithm that localizes predicate evaluation to the areas that satisfy the predicate, reduces runtime by over 50% to over 70%, and has similar accuracy to the Naïve baseline. Additionally, a localized predicate evaluation algorithm achieves near real-time performance in continuously evaluating the predicate over a sensor observation stream with a rate of 100,000 tuples/s, and achieves this runtime performance without sacrificing accuracy of the predicate result regions. The hypothesis is tested by implementing a prototype of the predicate evaluation stream query operator framework that contains the various, proposed localized algorithms for evaluating value predicates over ST fields and is assessed for runtime performance and accuracy of results.

## 5.5 Contributions

In order to evaluate value predicates over continuous phenomenon in near real-time, we propose the **STP-SQO framework** with the following contributions:

- We introduce the **STP-SQO framework** for efficiently and accurately evaluating predicates for high-throughput streams of sensor observations of ST continuous phenomena. The STP-SQO framework consists of several stream query operators that aim to interpolate the predicate accurately and at the same time limit the interpolation to the predicate result regions.

- We propose **two Region Growing Predicate Evaluation Algorithms** that restrict predicate evaluation to result regions surrounding stream tuples that satisfy the predicate. The **Breadth-First Region Growing ST Predicate Evaluation Algorithm** and **Scan Line Region Growing ST Predicate Evaluation Algorithm** start evaluation based on an initial set of grid cells that contain one or more tuples satisfying the predicate. Around each cell, the connected predicate result region is grown cell by cell using iteratively expanding search until completely surrounded by cells that do not satisfy the predicate.

- We present the **Tile-based Predicate Evaluation Algorithm** that partitions the prediction grid into tiles, each consisting of multiple grid cells, and performs predicate evaluation in batches using entire tiles. If a tile contains a tuple that satisfies the predicate, then all cells in the tile and all eight neighboring tiles are interpolated; thus, the tile approach is a greedy interpolation algorithm. The tile approach trades interpolating more cells for not having to keep track of a parallel, iteratively expanding search that is cell-based as done in the two region growing approaches.

- We introduce the **Phenomenon-Aware Predicate Evaluation Algorithm** as an incremental predicate evaluation strategy, i.e., an algorithm that 'learns' from the

past window result. The evaluation of a new query window starts with the result of the previous window and makes incremental updates to determine which cells will be elements of the predicate result, and which not. The Phenomenon-Aware interpolation operator exploits knowledge of the cells satisfying the predicate from previous window in order to determine the next set of cells to interpolate.

- To coordinate ST predicate evaluation operators, we introduce the **ST Predicate Region Evaluation Manager Operator** to coordinate and share state among parallel **ST Predicate Region Evaluator Operators**. The ST Predicate Region Evaluation Manager Operator keeps track of which cells have been queued and visited during expanding search (Region Growing algorithms), interpolated (Region Growing and Phenomenon-Aware algorithms), and classified as satisfying the predicate (Phenomenon-Aware algorithm).

## 5.6 Approach

In this section, we describe the operators and algorithms of the proposed ST Predicate Stream Query Operator Framework (STP-SQO framework). Pseudocode for the proposed operators and algorithms is located in Appendix C.

### 5.6.1 ST Predicate Stream Query Operator Framework Overview

The STP-SQO framework, shown in Figure 5.2, consists of the following operators:

- **Stream Split Operator**

  The Stream Split Operator splits and clones the incoming sensor update stream to two output queues to be consumed by two operators: the ST grid index and the *Seed Filter* operator. Since neither of these consumer operators modifies the tuples, tuples in the input stream are copied by reference to save memory.

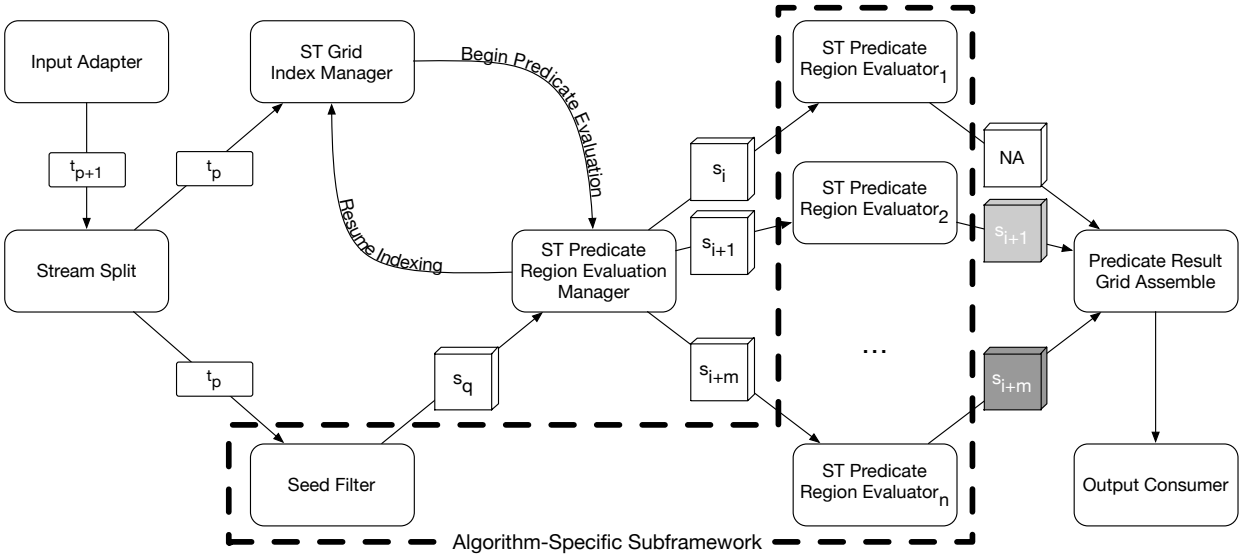



Figure 5.2. Stream operators for predicate evaluation

- **Seed Filter Operator**

  The Seed Filter Operator identifies regions likely to satisfy the predicate by filtering its input stream to select the set of tuples in a window that satisfy the predicate condition. This set of tuples that satisfies the predicate is then mapped to grid cells. The set of unique grid cells containing at least one tuple that satisfies the predicate is called the *seed cells*.

- **Sliding Spatio-Temporal Grid Index Manager Operator**

  The ST grid index Manager coordinates the indexing of all stream tuples and is described in Chapter 3. The ST grid index contains all the sensor observations and is shared between all parallel ST Predicate Region Evaluator Operator clones.

- **ST Predicate Region Evaluation Manager Operator**

  The ST Predicate Region Evaluation Manager operator coordinates the parallel ST Predicate Region Evaluator operators, keeps track of the regions being processed by each ST Predicate Region Evaluator operator clone, and coordinates access to the ST grid index.

- **ST Predicate Region Evaluator Operator**

  The *Predicate Region Evaluation Operator* uses the seed points to identify predicate regions to interpolate with ST interpolation using ST-IDW (see Chapter 3 for ST-IDW). Only interpolated cells which satisfy the predicate or where no data were found are put on the output queue.

- **Predicate Result Assemble Operator**

  The *Predicate Result Assemble Operator* sorts and arranges an output raster per window based on the interpolated tuples that satisfy the predicate. The raster contains all predicted values that satisfy the predicate, a code for all cells were no tuples were found within the configured interpolation radius, and a code for all cells where the predicted value does not satisfy the predicate.

### 5.6.2 Rationale for Predicate Evaluation

In order to justify an approach for predicate evaluation using ST interpolation, we discuss four alternative result types for evaluating predicate queries over ST field described in detail below.

1. **Predicate Result as Raw Seed Points**

   The simplest option is to evaluate the predicate on the raw stream of sensor observations. The tuples passing the predicate condition of the filter are termed *seed points*. If only the seed points are returned, then no information about the value of the phenomenon at ST points outside the seed points is provided to the user. It is left to the user to perform external analysis to assess the boundary and ST continuity of the phenomenon, a task which will prove difficult to do correctly as shown in options 2 and 3.

2. **Predicate Result based on Hull Constructed from Seed Points**

   Let us assume that the user is interested not in the individual seed points but instead the region satisfying the predicate. The bounding polygon of a set of points is called

a hull, that may either be a concave hull [111] or, more commonly, a convex hull [134]. A hull is created for a set of points, so if multiple regions satisfying the predicate are desired, the data points must first be partitioned using clustering [69]. A hull is then created for each cluster of points. With asynchronous, mobile sensors, creating a meaningful hull is a challenging problem unto itself. Assuming the set of seed points is not empty or equal to the set of observations in a window, a hull may over or underestimate the area of the phenomenon that satisfies the predicate and omit features including holes and separation between nearby regions. Additionally, a polygon of a hull captures the boundary of the set of seed points and does not capture values within the region.

3. **Predicate Result based on ST Snapshot Generated from Seed Points**
   The user may be interested in the field restricted to the region(s) satisfying the predicate, that is, a new field is returned instead of region objects as in option 2. However, issues arise if we attempt to represent the field using only the set of seed points. When the set of seed points is not equal to the set of points in a window, the estimated values at corresponding locations in representations either generated from the seed points or the entire points in a window may differ. Having only seed points means that there are not sample points available that show where the predicate is not satisfied. Performing ST interpolation from the seed points may overestimate the size region satisfying the predicate, by omitting holes and separation of regions, and interpolate values that differer from corresponding values calculated from the complete set of sample points.

4. **Predicate Result based on Complete ST Snapshot**
   The final approach is to use all observations for interpolation and apply the predicate to the interpolated result. Using all data points and ST interpolation means that more sample points are available to capture the ST behavior of the phenomenon

within a window and produce a snapshot at the IC. This alternative produces the most accurate representation of the predicate query result. The Naïve baseline uses this approach.

Thus, option 4, performing interpolation using the observation points, is necessary to most accurately capture the result of a predicate operator over the phenomenon. Using only the seed points for interpolation may omit sample points that help determine where holes and boundaries exist.

While option 4 produces the most accurate results, we seek a faster algorithm or algorithms than the Naïve baseline that produce the same accuracy.

### 5.6.3 Limiting Interpolation to Cells that Satisfy the Predicate

A value predicate applied to a geographic phenomenon typically retrieves features. For example, the predicate $p_{temperature>200}$ applied to a temperature field (above 200 °C) could identify fires within an observation region. Characteristically for such features, the sets of cells forming each feature in which the predicate evaluates to true are spatially and temporally contiguous. This fact can be exploited within an evaluation algorithm by identifying *seed cells*, i.e., grid cells having observations, which satisfy the predicate, and expanding seed cells to the surrounding region satisfying the predicate. By expanding seeds spatially and temporally, ST interpolation algorithms can potentially reduce the search for grid cells that need to be checked against the predicate and target interpolation to the cells likely to satisfy the predicate.

#### 5.6.3.1 Selection of Seed for Predicate Result Regions using a Seed Filter Operator

First, seeds of predicate result regions are identified by the *Seed Filter Operator*. It selects all tuples, which satisfy the predicate, and are termed *seed tuples*. Each seed tuple identified is mapped by location to its corresponding cell in the output grid, termed *seed*

*cell*. Since there may be multiple seed tuples that map to a single seed cell, in our proposed approach a hashmap keyed on the $(x, y)$ seed cell location is used to maintain the set of unique seed cells. The insertion order of cells into the set of seeds cells follows the random order of tuples in the stream.

Although tuples within a cell may meet the predicate condition, it is not clear whether the entire cell, once interpolated, will meet the predicate condition. Whether a cell is part of the predicate result can only be determined after using all available tuples in that cell, and potentially some surrounding cells, during ST interpolation. 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 creates seed 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.

The predicate evaluation algorithm used has implications for additional processing of seed cells to be performed by the Seed Filter operator. Thus, the Seed Filter operator is extended and specialized by each of the predicate evaluation algorithms presented in the following sections.

### 5.6.4 Seed-Based Predicate Evaluation Algorithm Alternative 1: Region Growing

The first proposed algorithms for expanding seed cells to full predicate results are two algorithms for region-growing. Seed-based region-growing algorithms are common for image segmentation in the domain of image processing, in which the values of grid cells are evaluated using a predicate [1, 42, 98, 150]. Similar to predicate evaluation, region-growing algorithms traverse a grid, starting from a set of seed points, and test cells based on their value. The difference for predicate evaluation over ST streams of continuous, dynamic phenomena, is that grid cells first need to be interpolated before the value can be tested

using the predicate. Further, all data are stored in main memory, processing is done in parallel, and performance demands are near real-time.

We will discuss two different region-growing algorithms we propose: Breadth-First (BF) and Scan Line (SL) and the processes of *seed selection* and *seed expansion*, performed by specializations of the Seed Filter Operator and ST Predicate Region Evaluator Operator, respectively.

**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 $\langle x, y \rangle$ 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 hashmap is used, allowing insertion of new coordinates in constant time, given an appropriate hash function.

**Seed Expansion:** Seeds are grown into regions of cells satisfying the predicate using a specialization of the ST Predicate Region Evaluator based on a breadth-first expansion. Each operator clone performs seed expansion in parallel as follows: it fetches a seed cell from the seed queue, and interpolates the seed cell's value based on all available samples points in the ST grid index for the cell. If the interpolated value of the seed cell satisfies the predicate, the cell is *expanded*. When a cell is *expanded*, its four neighboring cells (north, east, south, and west) are added to the seed's own "candidate cell" queue for this region. Before each cell is added to the queue, the operator checks the shared operator state, held by the ST Predicate Region Evaluation Manager, to determine whether the cell has already been queued for expansion in any of the other operator threads.

Region growing continues as long as the predicate is satisfied by the interpolated cell as shown in Figure 5.3. Next, the candidate cell queue is processed until empty. The cell at the head of the queue is fetched; the cell is interpolated, if it has not already

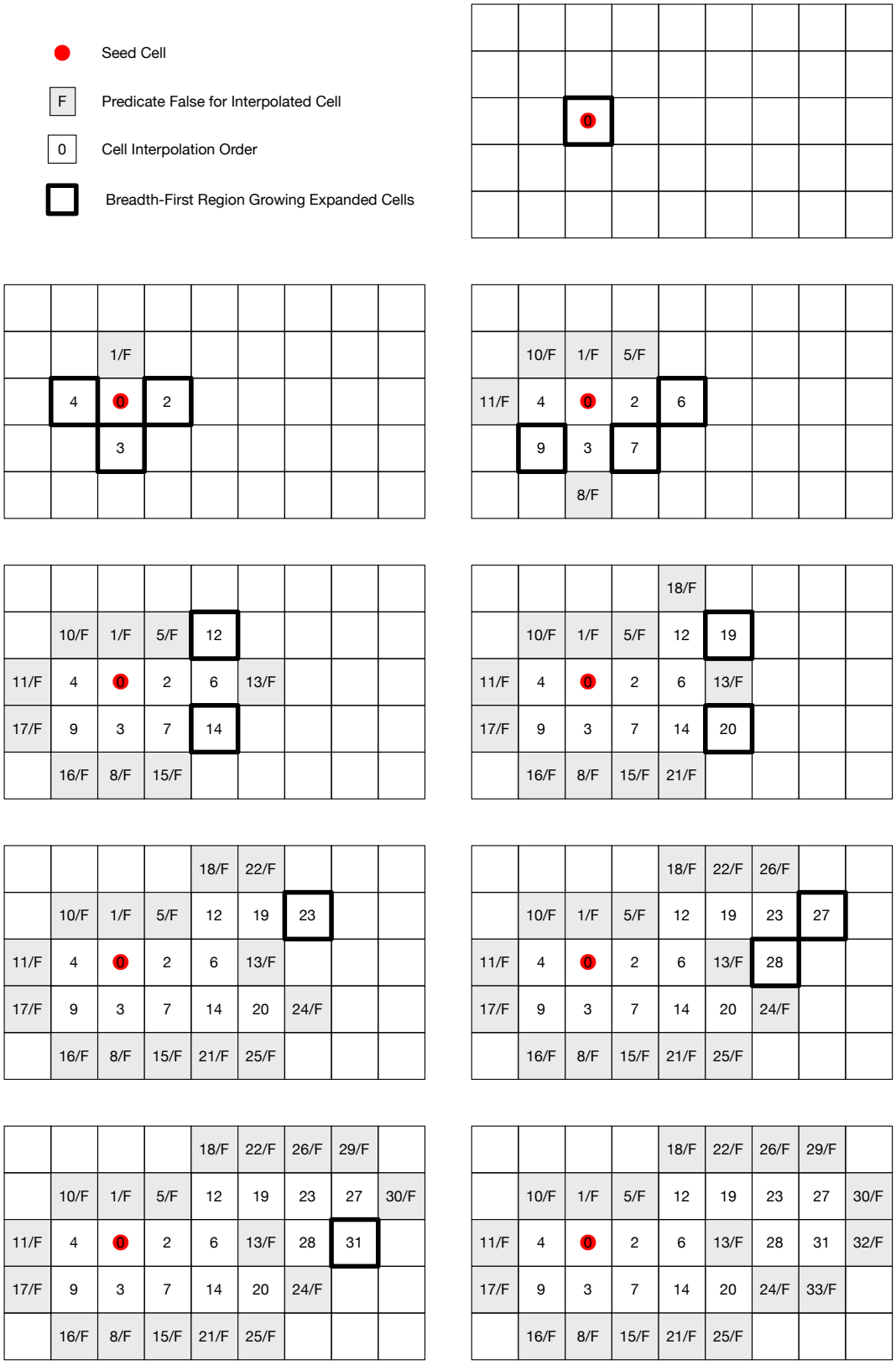



Figure 5.3. Breadth-First region growing algorithm

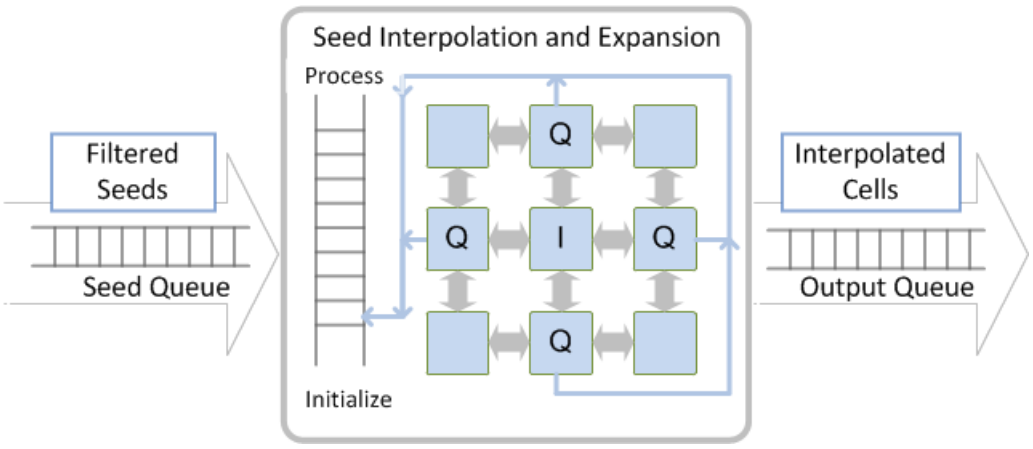



Figure 5.4. Breadth-First Region Growing ST Interpolation Operator

been by another operator clone; if it satisfied the predicate, it is expanded. The algorithm fetches the next seed once the internal candidate cell list is empty, and terminates when the queue of candidate cells and seeds are both empty.

The BF region growing algorithm is a breadth-first search without the termination condition of finding a particular item. Instead, the requirement is that the predicate be satisfied for a cell in order for its neighbors to be added to the queue. The traversal order of the BF region growing algorithm for a predicate result region is shown in Figure 5.3.

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 5.4).

#### 5.6.4.1 Scan Line Algorithm

The Scan Line (SL) algorithm is an alternative seed-based region growing algorithm; it can be visualized as starting from a seed cell and 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 BF (Section 5.6.4)

**Seed Expansion:** The basic idea of the SL is to grow regions line by line and in one direction at a time as shown in Figure 5.5. The algorithm for the Scan Line specialization of the ST Predicate Region Evaluator Operator proceeds as follows. First, fetch a seed from the seed queue and interpolate the value for the seed cell. If the seed satisfies the predicate, expand the region by interpolating the cell to the right of the seed. Continue to interpolate all cells in the current row to the right as long as the predicate continues to be satisfied. When the predicate is not satisfied, the boundary of the grid is reached, or a cell has already been marked as queued, switch to interpolating moving left in the current row beginning with the cell left of the seed cell. Once the set of contiguous set cells satisfying the predicate surrounding a seed in the current row have been interpolated, a *span* is generated to capture all cells to interpolate for the vertically adjacent rows. This span is one cell wider to both the left and right than the set of contiguous cells from the current row. The spans are then added to a span queue to iteratively expand the search vertically as well as to wrap around into concavities of a result region. 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 both the right and left limits of the row, and then again jumps vertically. This is repeated until the connected set of cells satisfying a predicate around a seed is is fully interpolated. Compared to the basic BF algorithm, which searches in four directions, we mark the cells satisfying the threshold by traversing line sections, focusing most effort in searching to the right rather than equally in four directions.

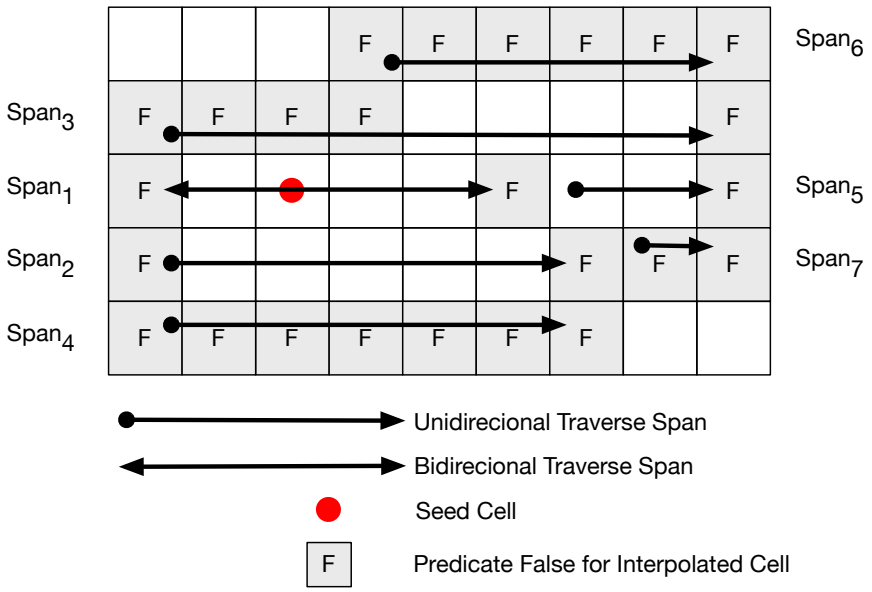



Figure 5.5. Scan Line Spans During Region Growing for a Concave Region

### 5.6.5 Seed-Based Predicate Evaluation Algorithm Alternative 2: Tile Expansion

If a grid is very large or has a high resolution, using a breadth-first traversal for region growing to identify cells satisfying the predicate, performed in parallel by multiple operator clones expanding sometimes competing, overlapping regions, 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., $2 \times 2$ or $4 \times 4$ 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 evaluate all (and likely more) grid cells that, after interpolation, will be an element of the predicate result. Tile expansion is a rough, but fast estimate of such cells. At some point, interpolating all cells is more efficient than a tile-based approach.

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, queuing the eight neighboring tiles for interpolation. Another consideration is *tile size*. If the tile size is large, the percentage of irrelevant cells being interpolated likely is larger. Choosing a small tile 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 factor of 16 (power of 2) is advantageous (e.g., 1, 2, 4, 8 or 16) as a whole number of tiles fit within the radius. If an interpolation radius $r = 17$ were used instead, choosing tile size 16 would mean that a buffer of 2 tiles around each seed tile would be needed to guarantee that all cells were interpolated, interpolating all cells within a buffer of 32 cells around the tile instead of the needed 17 cells.

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 hash map 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 algorithm 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 queuing 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 to be processed in parallel by the ST Predicate Region Evaluators. In all cases the complexity for this is $O(x * (\frac{r}{s})^2)$ where $x$ is the number of expanded tiles, $r$ is the interpolation radius, and $s$ is the number of tiles in a distance of $r$. In the worst case, where $r = 1$, $s = 1$, and the grid contains $m$ cells and each tile is $1 \times 1$, the complexity is $O(m)$.

For the tile based approach, seed selection, expansion of seed tiles, and the conversion of expanded tiles to cells are all handled in the Seed Filter. All cells in the expanded seed tiles are evaluated whether or not they satisfy the predicate using the ST Predicate Region Evaluator.

### 5.6.6 Discussion of Seed-Based Approaches

Overall, the performance of the proposed algorithms depends on the following aspects: 1) indexing of raw tuples, 2) filtering, 3) seed selection, and 4) seed expansion with interpolation (either by region growing or tile expansion), and 5) interpolation. The cost for indexing raw tuples and filtering is $O(n)$ where $n$ is the 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 BF, all four neighbors are checked for each cell. In SL, 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 BF and SL 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. For SL, only one-quarter of the edges are traversed. The space complexity for both algorithms is $O(|V|)$, where $|V| = mn$ is the number of cells in the grid. Overall, the cost for each algorithm is determined by the interpolation cost and grid size, and depends linearly on the number of tuples and number of cells in the grid.

## 5.7 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 $w_{i-1}$ during evaluation of the current window $w_i$ 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 [57], 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.

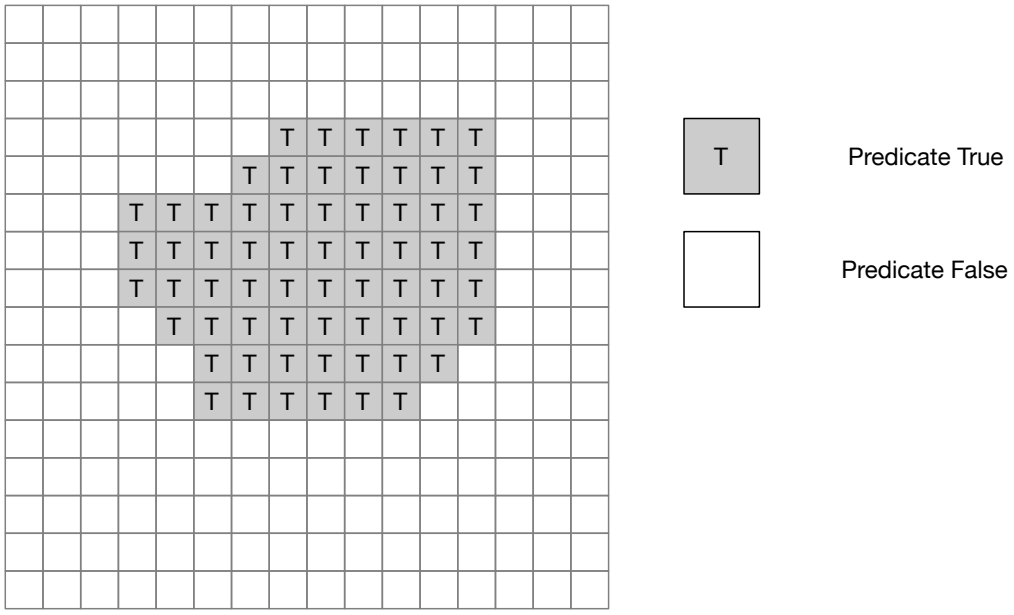



Figure 5.6. Classified cells satisfying the predicate from query window $w_{i-1}$

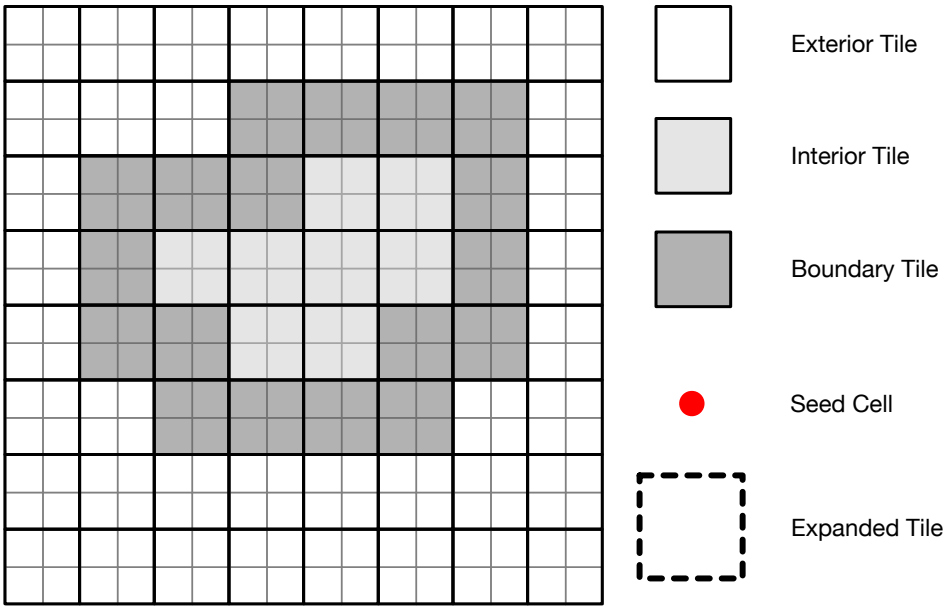



Figure 5.7. Classified tiles satisfying the predicate from query window $w_{i-1}$

### 5.7.1 Rationale

The Phenomenon-Aware predicate evaluation approach is an extension of the Tile-based algorithm (Section 5.6.5), classifying the predicate result from query window $w_{i-1}$ by tile. Tiles are classified based on whether they contain the `Interior`, `Boundary`, or `Exterior` of the result from $w_{i-1}$. In essence, the objective of tile classification is to minimize tile expansion in $w_i$ for areas that satisfied the predicate in $w_{i-1}$. An example of a classified predicate query result by tile for $w_{i-1}$ is shown in Figure 5.7 and depicts the following classes of tiles:

**Interior:** An `Interior` tile contains only cells that fulfilled the predicate in $w_{i-1}$. `Interior` tiles contain cells that are most likely also part of the predicate result for window $w_i$.

**Boundary:** A `Boundary` tile contains both cells that fulfilled the predicate during $w_{i-1}$ and cells that did not fulfill the predicate in the previous window. `Boundary` tiles capture areas that are most likely to change from query window $w_{i-1}$ to $w_i$.

**Exterior:** Tiles that contain only cells that did not fulfill the predicate in $w_{i-1}$ are termed `Exterior` tiles. `Exterior` tiles might still be of the same types in window $w_i$, 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. 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-interpolates updated values for cells that satisfy the predicate. If small tile sizes are used, e.g. $1 \times 1$, an additional step is needed that reclassifies all `Interior` tiles that are adjacent to an `Exterior` tile as `Boundary` tiles.

### 5.7.2 Tile Classification and Expansion Overview

The process of classification of the result from $w_{i-1}$ and expansion for $w_i$ is depicted in Figure 5.8. The figures shows `Interior` tile A, `Exterior` tile B, and `Boundary` tile C. The process of expansion for each tile is summarized below.

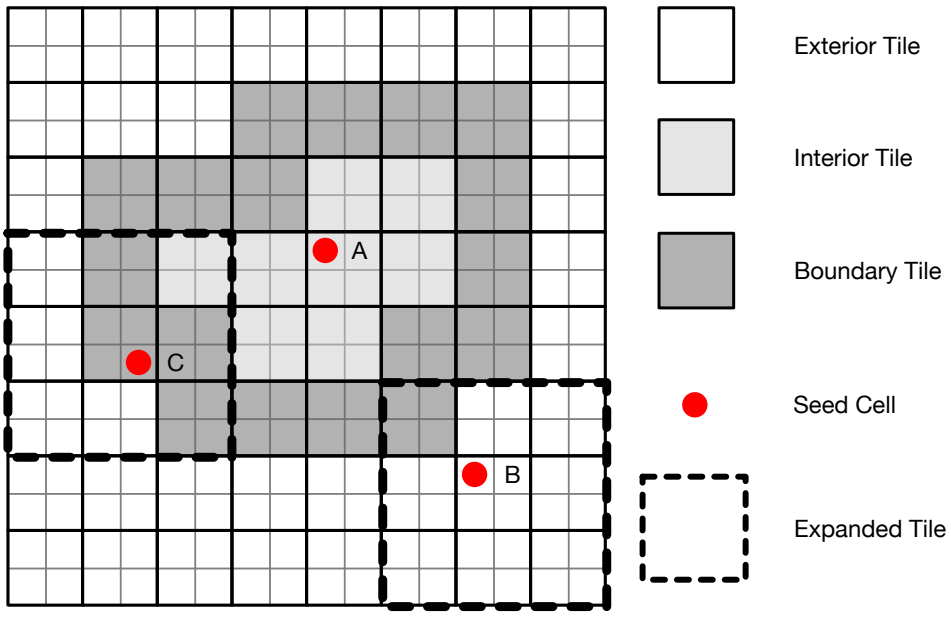



Figure 5.8. Expanded tiles around seeds in query window $w_i$

**Interior** e.g., tile A

- All cells in the tile satisfied the predicate in $w_{i-1}$
- Do not expand if there is a new seed for $w_i$

**Exterior** e.g., tile B

- No cells in the tile satisfied the predicate in $w_{i-1}$
- Expand if there is a new seed for $w_i$

**Boundary** e.g., tile C

- Contains at least one cell that did and once cell that did not satisfy predicate in $w_{i-1}$
- Expand if there is a new seed for $w_i$

### 5.7.3 Classifying Tiles

The Phenomenon-Aware predicate evaluation algorithm receives the following input: a) a queue of all interpolated cells of window $w_{i-1}$, 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 $w_i$.

**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 $w_{i-1}$. 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` or the cell satisfies the predicate and falls into a tile already classified as `Exterior`, the tile is

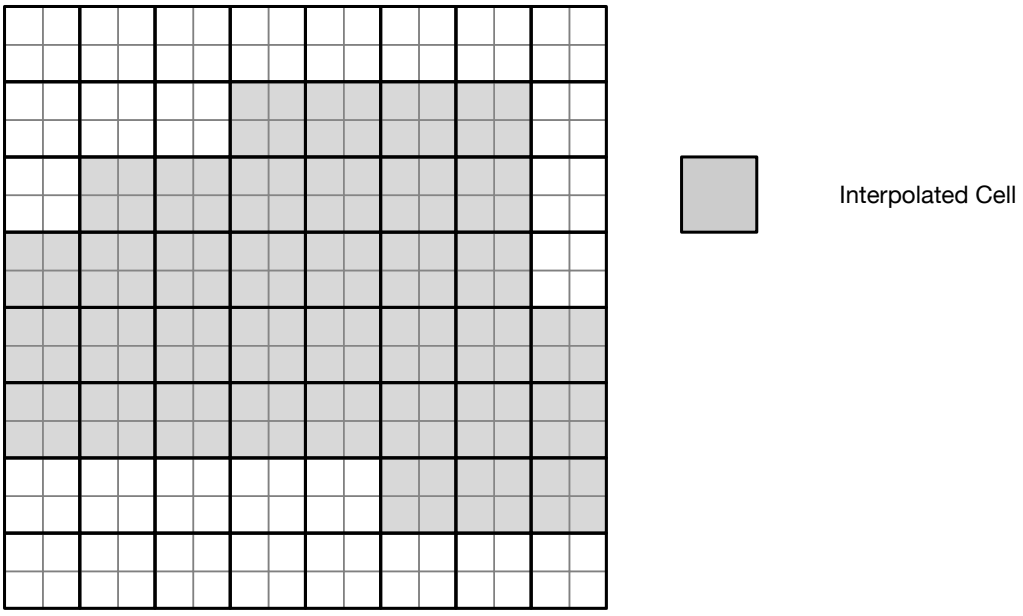



Figure 5.9. Cells to perform predicate evaluation on for query window $w_i$

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 5.7).

### 5.7.4 Detailed Expansion Procedure of Tiles by Types

In the next step, the queue of new seed tuples, which arrived during $w_i$ is processed, and the tiles are reclassified (see Figure 5.8, 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 5.8, tile A). If a new seed is located within a `Boundary` tile, the `Boundary` tile is marked for expansion (Figure 5.8, tile C). The new seed indicates that the predicate result region may have 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 5.8, tile B). If a seed falls into a `No Data` tile, the tile is reclassified as `Exterior`, and marked for expansion as `Exterior/Expand`.

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. The cells to be interpolated are shown in Figure 5.9.

## 5.8 Performance Evaluation

In this section, we evaluate the performance of the different proposed algorithms for predicate evaluation, and compare them to the baseline naïve approach.

### 5.8.1 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 (roughly one third of Japan around the 2011 Fukushima nuclear incident). The discussed algorithms and operators of the STP-SQO framework were implemented as stream query operators assuming a DSE environment. Test are performed using the same data sets as described in Section 4.2.1. The accuracy of the tested configurations is evaluated in Section 5.8.3 and the runtime of the configurations is evaluated in the remainder of the section.

#### 5.8.1.1 Implementation and Run-time 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 (6 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. 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:
$Runtime = Slowest(AVG(time_{index}), AVG(time_{seed})) + AVG(Slowest(IDW_1 \dots IDW_8))$.

### 5.8.2 Parameter Selection

For the following tests a sliding window with a length and slide of 4 ticks, referred to as a tumbling window, was used. Tests were run with ten iterations of five consecutive windows over both the Cambridge and Japan simulated sensor observation stream data sets described in Section 4.2.1. Sensor populations between 32K and 256K were used and each sensor had a 25% probability of updating for a particular tick (i.e., on average each sensor updates at one point in the window). A ST snapshot for each window was constructed using a $512 \times 512$ grid and an IC at the end of the query window. ST interpolation was performed using the ST Shell Approach (ST Shell) and ST Adaptive $k$-Shell (ST a$k$-Shell) algorithms for ST-IDW described in detail in Section 3.6.4. A radius $r = 16$ was used for ST Shell and ST a$k$-Shell and for ST a$k$-Shell $k = 4$ was used. The size of the predicate result area was quantified as the proportion of number of cells of predicate result compared to number of cells of the entire observation region. Snapshots from three points in the ST evolution of the phenomenon, with corresponding predicate result areas small (10%), medium (28%), and large (38%), were used to assess the performance of the algorithms.

### 5.8.3 Accuracy of Predicate Result Regions

In order to investigate the accuracy of each predicate algorithm, the set of cells satisfying the predicate in the interpolated result is compared with the cells that satisfy the

predicate in the ground truth. This dissimilarity between the two sets is calculated using the Jaccard distance. The Jaccard distance (shown in Equation 5.1) is related to the Jaccard index [74], which computes the similarity of two sets by dividing the number of cells in the intersection by the number of cells in the union of the two sets (shown in Equation 5.2). Jaccard distance values range between 0 and 1 with 0 indicating that the two sets are equal (most similar) and 1 that the two sets are disjoint (least similar).

$$D_j(A, B) = 1 - J(A, B) \tag{5.1}$$

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{5.2}$$

The Jaccard distance was computed for for all tested configurations and is shown for the Cambridge (see Figure 5.10) and Japan (see Figure 5.11) data sets for the medium and large phenomenon sizes. The results show that accuracy of the detected area improves, shown by a decrease in the Jaccard distance, as the size of the predicate result area increases. For the same phenomenon size, the predicate results for the Cambridge street network were more accurate than the Japan street network, aided by the more even distribution of sensors over space in the Cambridge data set. For the medium sized predicate result using ST a$k$-Shell resulted in better accuracy than using ST Shell while for the large predicate result ST Shell had marginally better accuracy than ST a$k$-Shell. Since the Region Growing, Tile-based, and Naïve are all guaranteed to evaluate the entire predicate region, the variation in their accuracy is depended on the random subset of sample points in each window. The the results of the Phenomenon-Aware approach lie within this variation, confirming that none of the algorithms sacrifice accuracy for the configurations and data sets tested.

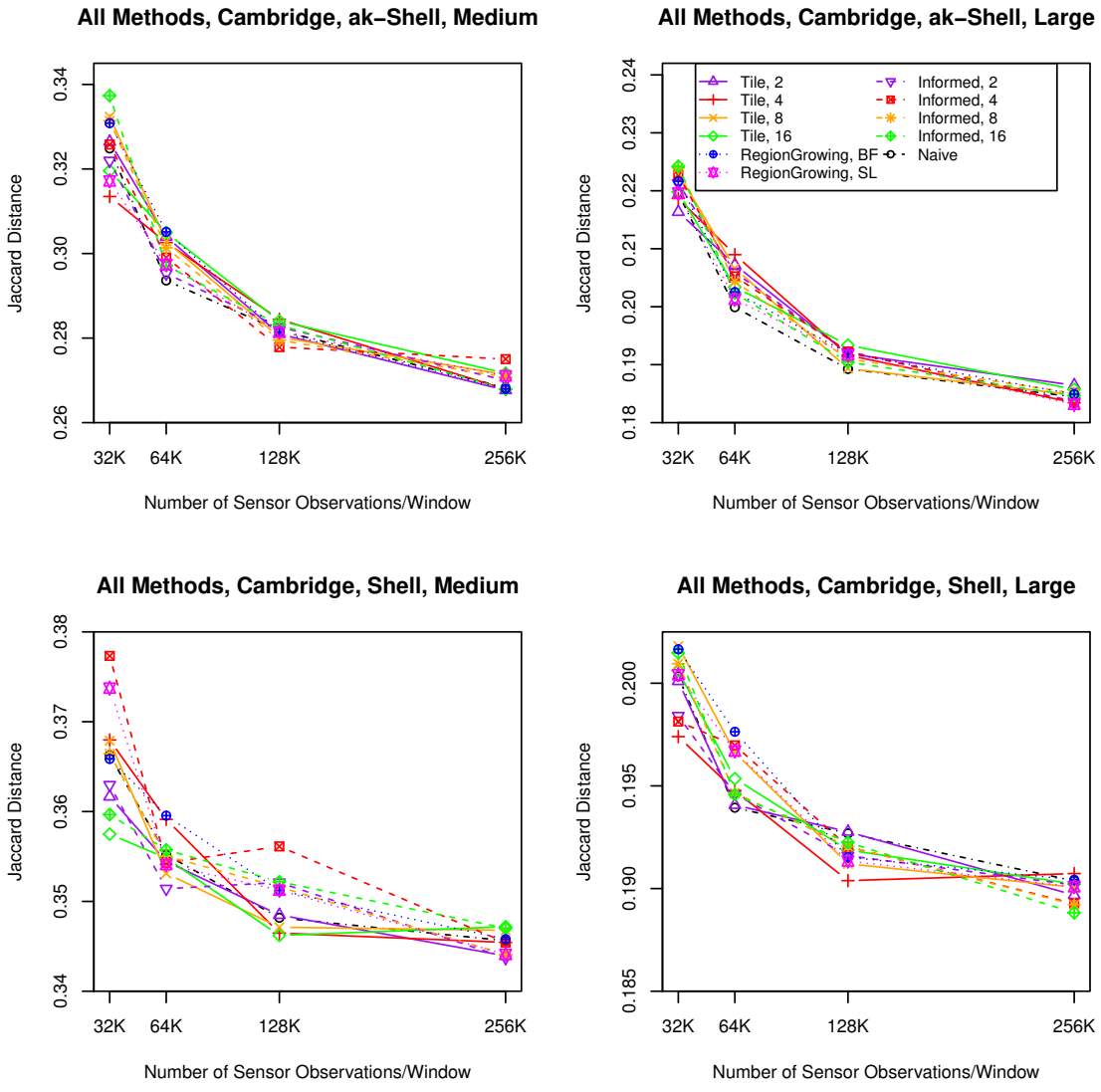



Figure 5.10. Predicate result accuracy summary: Cambridge data set with medium and large predicate result sizes

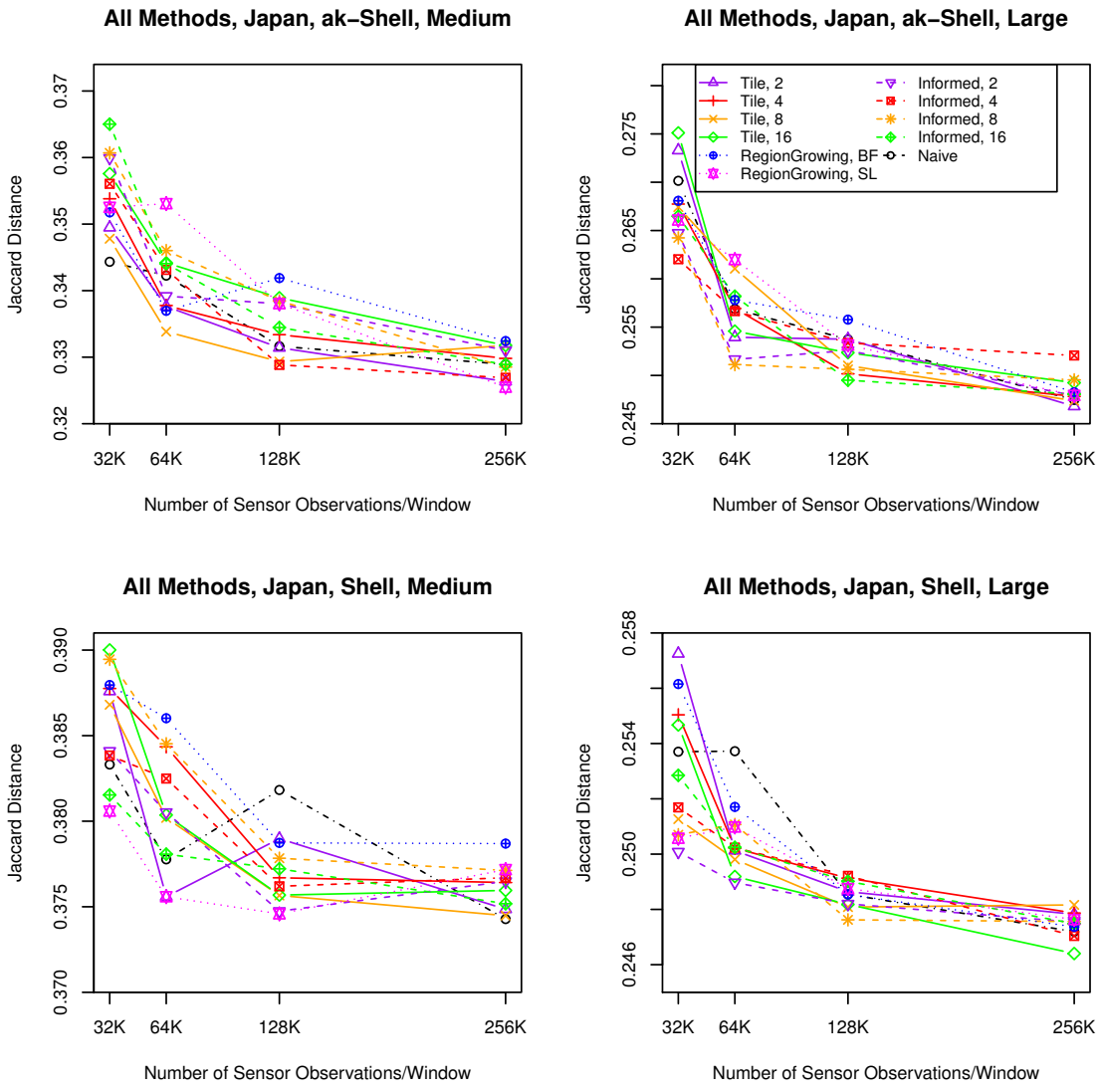



Figure 5.11. Predicate result accuracy summary: Japan with medium and large predicate result sizes

### 5.8.4 Naïve vs. Region Growing Algorithms

First, we compare the performance of the Naïve method to the region growing algorithms, i.e., BF and SL (Figures 5.12 and 5.13). In the Naïve method all cells of the entire observation region are interpolated and predicate evaluation is performed after interpolation. The region growing algorithms start by filtering tuples and expanding seed cells. We tested the Naïve and region growing algorithms in combination with the ST a$k$-Shell (Figure 5.12) and ST Shell algorithms (lower 3 graphs) for ST-IDW interpolation using the three different phenomenon sizes (small, medium, large). The results for ST Shell 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 ST a$k$-Shell algorithm. This is due to exploiting the high-speed caching as a linear traversal during interpolation results in most of 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 ST a$k$-Shell. We also observe that the runtime initially decreases with increasing number of observations since the search can be reduced to fewer cells in ST a$k$-Shell. However, the performance decreased again once the set of observation reaches about 128K samples/window as expected. Comparing ST a$k$-Shell and ST Shell, the ST a$k$-Shell version is significantly faster than ST Shell (e.g., runtime between 0.14-0.2s for ST a$k$-Shell/SL, medium result size compared to 0.8-2s for ST Shell/SL, medium result size).

### 5.8.5 Tile Expansion

We tested the performance behavior of the Tile-based algorithm. This algorithm 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

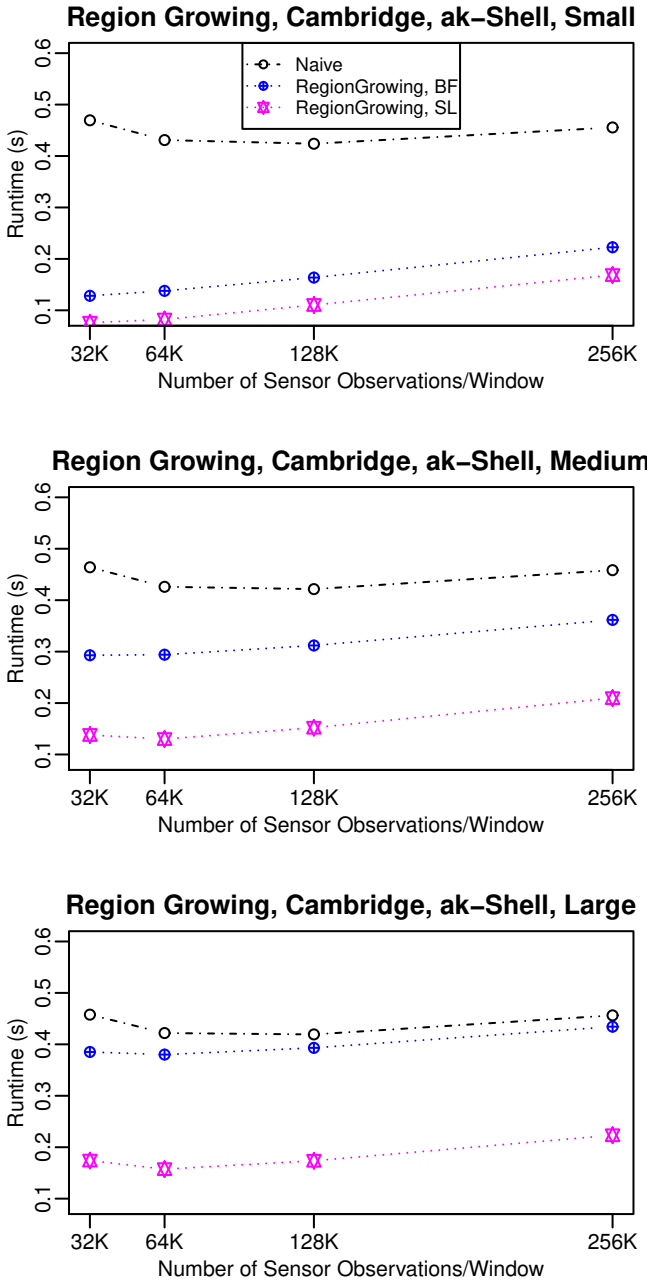



Figure 5.12. Region growing using ST a$k$-Shell runtime

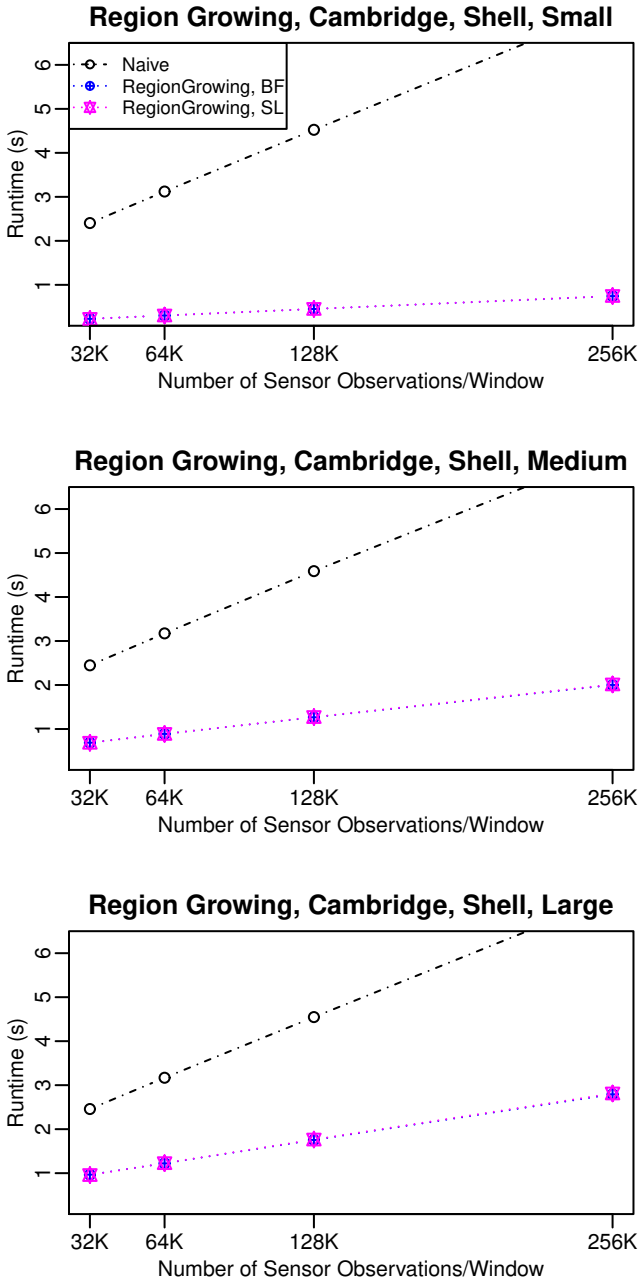



Figure 5.13. Region growing using ST Shell runtime

$S_1 = 2 \times 2$, $S_2 = 4 \times 4$, $S_3 = 8 \times 8$, and $S_4 = 16 \times 16$. The tile expansion approach only uses the seeds from the current window, and is thus *uninformed* of the previous state of the phenomenon, while the Phenomenon-Aware algorithm (Section 5.7) is *informed* about the extent of the phenomenon gleaned from the last interpolation step (the comparison is discussed in Section 5.8.6). The results for tile size impact are shown in Figure 5.14. As expected, the Tile-based 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 4 is the best choice for consistently good performance for ST a$k$-Shell for both Cambridge and Japan and all three phenomenon sizes.

### 5.8.6 Runtime Comparison of all Predicate Evaluation Algorithms

Figure 5.15 shows a comparison of all tested predicate evaluation algorithms: Region growing (BF and SL), Tile-based (uninformed), Phenomenon-Aware (informed), 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 tile sizes are dashed lines, while the uninformed tiles method results are solid lines. We show results for the Cambridge and Japan data sets. The difference between the data sets 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 ST Shell tests (lower two graphs of Figure 5.15), results confirm that the Naïve method has the longest runtime, while the region growing methods have the best performance compared to the tile-based and the Naïve approaches, over both 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. A tile size of $2 \times 2$ was the fastest for the small predicate area and $4 \times 4$ was fastest for the medium and large predicate areas. For ST a$k$-Shell (top two graphs of Figure 5.15), we can see that SL

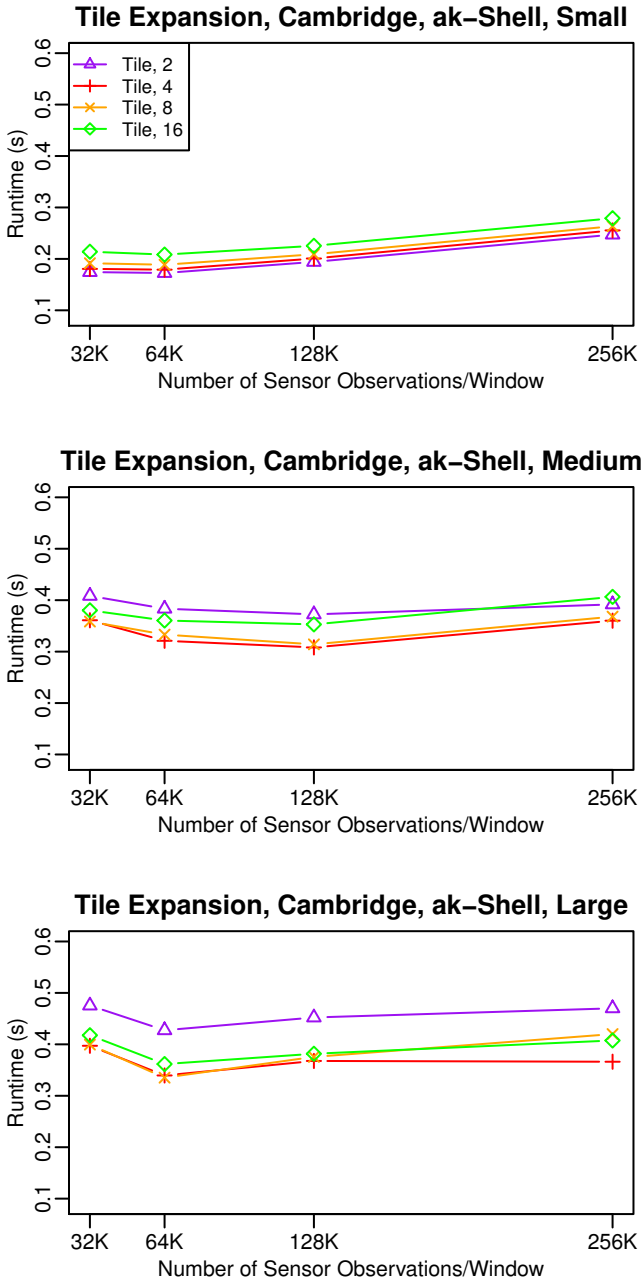



Figure 5.14. Tile-based expansion

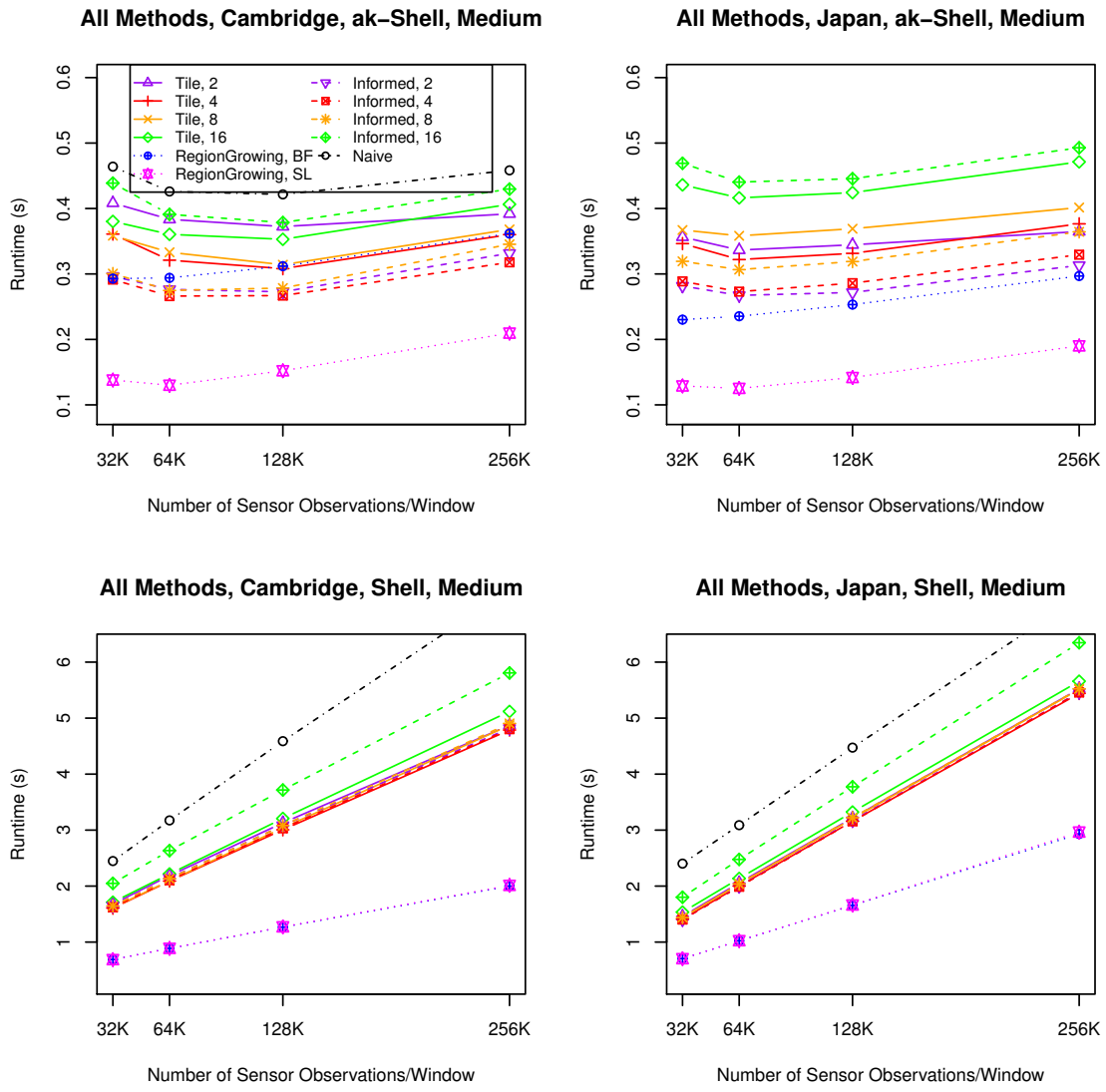



Figure 5.15. Runtime comparison of all predicate evaluation algorithms

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 ST fields is beneficial using the Phenomenon-Aware over the Tile-based algorithm. However, using SL in combination with ST a$k$-Shell resulted in the lowest runtime for the configurations tested. SL is advantageous as by performing region growing line by line and in one direction at a time,

using a linear traversal of the ST grid and exploiting locality while still guaranteeing that the entire predicate result region will be interpolated and accurately extracted.

## 5.9 Summary

In this chapter, we proposed and evaluated the STP-SQO framework to efficiently and accurately evaluate value predicates over ST fields observed by over 250,000 asynchronous, mobile sensors. The STP-SQO framework 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 proposed different seed expansion algorithms (BF and SL region growing, and Tile-based expansion) and explored the impact of using the knowledge of the previous query window result in the Phenomenon-Aware algorithm. Our analysis and performance results show that the SL region growing algorithm outperforms all other tested algorithms regardless of data size or specific characteristics of the data set.

The Phenomenon-aware tile expansion algorithm performed better than the (uninformed) Tile-based approach for using tile sizes $2 \times 2$, $4 \times 4$, and $8 \times 8$ for the medium-sided predicate area. The predicate results were validated by comparing the output of each method to the baseline Naïve approach. Assessment of the accuracy of the predicate result snapshots shows that while all tested predicate evaluation algorithms perform faster than the Naïve approach, this increase in performance comes without a trade-off in accuracy over the baseline Naïve approach. Though the SL was the fastest among the proposed predicate evaluation algorithms for the predicate result regions tested, additional analysis is needed to determine the best predicate evaluation algorithm for even larger predicate result sizes (e.g., 50%, 75%, and 100% of the raster) and shapes of the predicate result regions.

The STP-SQO framework is able to efficiently evaluate value predicates in near real-time on ST fields observed by massive numbers of sensor observation streams and represented using ST snapshots.

# CHAPTER 6
# CONCLUSIONS AND FUTURE WORK

In this dissertation we have investigated the problem of providing support for near real-time representation and analysis of continuous ST phenomena observed by large numbers of mobile and asynchronous sensors for continuous queries using a DSE.

## 6.1 Contributions and Major Findings

We have proposed and evaluated two stream query operator frameworks: the STI-SQO framework for representing and the STP-SQO framework for analyzing ST fields. The frameworks provide scalable, stream-based query operators for use in a DSE for near real-time representation and analysis of continuous phenomena observed by massive numbers of asynchronously sampling, mobile sensors. Additionally, the frameworks demonstrate the suitability of DSEs, extended with stream-based algorithms for geospatial operations on fields, for scenarios requiring near real-time representation and analysis of ST phenomena as we will explain in the major findings in Section 6.1.2. This section highlights the contributions and discusses the primary findings of this thesis.

### 6.1.1 Contributions

The following are the contributions of this thesis:

- **ST Interpolation Stream Query Operator Framework**

  We proposed the scalable ST Interpolation Stream Query Operator Framework (STI-SQO framework) that transforms a high-throughput stream of sensor observations into representations of continuous spatio-temporal phenomena in near real-time. The transformation of sensor observation streams in a query window into ST snapshots is performed as a stream query that is evaluated via a framework of tightly-collaborating, novel stream query operators.

- **Sliding Spatio-Temporal Grid Index**

  A key component of STI-SQO framework is an in-memory, Sliding Spatio-Temporal Grid Index (ST grid index), conceptually visualized as an ST cuboid, that enables efficient search in space and time for tuples during ST interpolation. The ST grid index supports an isotropic view of space and time facilitated by an *ST anisotropy ratio*, the introduction of *Isotropic Time Cells (ITCs)*, and a mapping between ITCs and *time blocks* through the use of *time subblocks*. The ITCs are centered about the Interpolation Center (IC), enabling the ability to search, emanating from the center of each raster cell, for proximate observations in space and time. Additionally, the ST grid index is designed to be efficiently reused between windows, able to quickly identify the outdated portion of the ST grid index by using time blocks and clear tuples so that portion may be refilled with the newest data added to the window.

- **ST Interpolation Algorithms**

  For the stream query operator framework, two ST-IDW algorithms for near real-time *ST Interpolation Operators* with a throughput of over 100,000 tuples/s are introduced. The *ST Shell* algorithm uses *all* observations within a spatio-temporal shell around each cell in the output grid and within the temporal bounds of the window. In the *ST ak-Shell* method, the observations used for a particular predicated cell are determined using an iteratively expanding ST search for a lower bound of $k$ points within a maximum ST radius $r$.

  The ST-IDW algorithms support any arrangement of sensor observations in a ST point cloud, allowing both asynchronous and mobile sensors. The reusable *Cylindrical Shell Template* and *Nested Shell Template*, for ST Shell and ST a$k$-Shell, respectively, are introduced as templates for searching the index for nearby tuples to each cell interpolated. We introduce an ST anisotropy ratio to merge spatial distance and temporal difference between and into a combined ST distance weight.

- **Data Skew Adaptive kNN-based ST-IDW Interpolation**

  The deployment of massive numbers of sensors, many of them moving, can lead to data skew over space and time. To deal with this type of skew adaptively, we introduce the dynamic ST a$k$-Shell operator. This operator locates a bounded number ($k$) of sample points within a bounded search radius ($r$) during an iteratively expanding ST search using the ST grid index and Nested Shell Template (NST) that terminates when $k$ is reached. The ST a$k$-Shell operator is able to adapt to data density, with $k$ serving as the termination condition when data are dense and $r$ when data are sparse.

- **Support for Flexible Query Result Presentations for Stream Queries over ST Continuous Phenomena**

  We introduced the concept of the *IC* as a user-defined time stamp within a query window at which a ST snapshot of the continuous phenomenon is constructed using ST interpolation. The snapshot represents the predicted state of the phenomenon at that instant based on the observations from the entire window that are in spatial and temporal proximity to each prediction location. Thus, the user can control the time instant at which a ST snapshot is generated within each window. Further, we introduced two types of stream query result representations for ST continuous phenomena, that are either a single summary ST *snapshot* at the IC or a *window movie* composed of multiple snapshots showing an animation of the ST phenomena over a window.

- **Stream Query Operator Framework for Evaluating Value Predicates over ST Phenomena**

  We introduce ST Predicate Stream Query Operator Framework (STP-SQO framework) with different algorithms for evaluating value predicate queries over the ST phenomena accurately in near real-time. This novel approach evaluates the

predicates over the representation of a continuous phenomenon and returns result regions instead of filtering the raw sensor data stream and returning raw observations. Overall, we introduce and evaluated three different approaches for predicate evaluation: *Region Growing Predicate Evaluation*, *Tile-based Predicate Evaluation*, and *Phenomenon-Aware Predicate Evaluation* performed in parallel by the *ST Predicate Region Evaluator Operators*.

- **Region Growing ST Predicate Evaluation Algorithms**

  We proposed **two algorithms for Region Growing Predicate Evaluation**, the *Breadth-First Region Growing ST Predicate Evaluation Algorithm* and *Scan Line Region Growing ST Predicate Evaluation Algorithm*, that restrict predicate evaluation to result regions surrounding the grid cells containing stream tuples that satisfy the predicate, called seed cells. Around each seed cell, the connected predicate result region is grown cell by cell using iteratively expanding search until completely surrounded by cells that do not satisfy the predicate. The Breadth-First Region Growing algorithm grows predicate result regions using a breadth-first traversal and the Scan Line Region Growing algorithm expands predicate result regions one line at at time.

- **Tile-based Predicate Evaluation Algorithm**

  We presented the *Tile-based Predicate Evaluation Algorithm* that partitions the prediction grid into tiles, each consisting of multiple grid cells, and performs predicate evaluation in batches using entire tiles. If a tile contains a tuple that satisfies the predicate, then all cells in the tile and all eight neighboring tiles are interpolated; thus, the tile approach is a greedy interpolation algorithm. The tile approach trades interpolating more cells for not having to keep track of a parallel, iteratively expanding search that is cell-based as done in the two region growing approaches.

- **Phenomenon-Aware Predicate Evaluation Algorithm**

  We introduced the *Phenomenon-Aware Predicate Evaluation Algorithm* as an incremental predicate evaluation strategy, i.e. an algorithm that 'learns' from the past window result. The evaluation of a new query window starts with classifying the result of the previous window by tile and makes incremental updates to determine which cells will be elements of the predicate result, and which not. The Phenomenon-Aware interpolation operator exploits knowledge of the cells satisfying the predicate from previous window in order to determine the next set of cells to interpolate.

- **ST Predicate Region Evaluation Manager Operator**

  To coordinate ST predicate evaluation operators, we introduced the *ST Predicate Region Evaluation Manager Operator* to coordinate the parallel ST Predicate Region Evaluator Operators. The ST Predicate Region Evaluation Manager Operator keeps track of which cells have been queued and visited during expanding search (Region Growing algorithms), interpolated (Region Growing and Phenomenon-Aware algorithms), and classified as satisfying the predicate(Phenomenon-Aware algorithm).

### 6.1.2 Major Findings

The hypotheses posed in Chapter 3 and Chapter 5 were verified though evaluation of a prototype implementation of the proposed stream query operator framework and assessed with regard to accuracy and runtime performance using real-world and simulated data streams and confirmed with the the following findings.

- Using the ST a$k$-Shell algorithm, STI-SQO framework was able to generate $500 \times 500$ sized ST snapshots in less than 2.5 seconds with an NRMSE under 0.19 (19%) for over 250,000 observation from mobile, asynchronous sensors on an urban street network. In reconstructing a dynamic,ST phenomenon from sample points using interpolation, some error between the predicted and observed behavior is expected; a

lower is preferable as it means the prediction is more accurate. Thus, the proposed ST Interpolation Stream Query Operator Framework achieves the processing throughput target of 100,000 tuples/s and demonstrated the suitability of DSEs for near real-time representation of continuous ST phenomena with appropriate stream query operator framework support.

- Testing the introduced ST Predicate Stream Query Operator Framework showed that it analyzed value predicates over ST fields accurately, compared to the baseline approach. Using the Scan Line algorithm for the ST Predicate Region Evaluator Operator with in the ST Predicate Stream Query Operator Framework and using ST a$k$-Shell based ST-IDW interpolation, predicate evaluation takes around 0.2 seconds for both urban and rural street networks. Hereby, 250,000 sensor observations per window from mobile, asynchronous sensors were used; predicate results took up on average 28% of a result raster that had a resolution over $500 \times 500$ cells. The Scan Line algorithm, which grows regions from seed cells line by line, was the fastest predicate evaluation algorithm for all tested configurations. These results confirm that the STP-SQO framework is able to accurately and efficiently evaluate value predicates on ST fields observed by massive numbers of sensor observation streams and represented using ST snapshots.

- The performance of these frameworks demonstrates the suitability of DSEs for efficiently and accurately representing and analyzing continuous phenomena observed by over 250,000 asynchronous, mobile sensor streams in near real-time using the STI-SQO framework and STP-SQO framework.

- While this thesis investigated the configuration of the STI-SQO framework parameters $r$, $k$, $p$, $a$, and IC for the ST Shell and ST a$k$-Shell stream-based algorithms for ST-IDW using different sampling arrangements and numbers of sensors, in a real world setting, both the ST phenomenon and ST distribution of

sensor observations may change over time, depending on the phenomenon. As such, the parameters $r$, $k$, $p$, and $a$ of the STI-SQO framework need to be adapted to meet both runtime and accuracy requirements. This leads to future work of how these parameter can automatically be determined on-the-fly during the continuous stream query. Similarly, methods need to be included that assess the accuracy of the output rasters continuously.

- This thesis investigated stream-based predicate evaluation algorithms for value predicates over ST fields represented as raster snapshots, and predicate results sized less than 40% of the area were investigated. For predicates covering a large area of the output, e.g. 95%, it is expected that SL will perform slower than the Naïve algorithm as the results showed that the difference in time between SL and Naïve diminished as the predicate result size increased. The size of a predicate result may change over time during a stream query, meaning that the optimal predicate evaluation algorithm needs to be continuously adapted.

## 6.2 Future Work

This thesis presented stream query operator frameworks for representing ST fields and evaluating predicates over the values of ST fields in real-time. This work has led to the discovery of several opportunities for future research directions.

### 6.2.1 Automatic Accuracy Assessment and Parameter Optimization

In an emergency scenario, one would desire to have the most accurate and timely information in order to best respond. This means that the transformation of ST sensor observation streams into ST representations, and analysis of ST fields, needs to meet both application-specific accuracy and real-time constraints. While in the experiments conducted in Chapter 4 and Chapter 5 the ground truth was known, in a real-world setting the ground truth is often unknown and only sensor observations are available.

#### 6.2.1.1 Automatic Accuracy Assessment

For a real-time sensing environment, a method to automatically assess the accuracy of the representation using only the sensor observations is necessary. Cross validation is one technique to address this question. To perform cross validation, a portion of the sensor observations in a window are set aside to be used at the end of a query window for validation and the remaining sample points are used for ST interpolation. Then the difference between the predicted values from interpolation and the values of the validation points are calculated. Since the observed points do not line up directly with raster cells or the IC, a naïve option to address this is to calculate a value for each validation point using the same ST interpolation parameters.

Though this option seems intuitive, issues assessing the accuracy can arise if the distribution of sensor observations used for validation is not representative of the prediction locations (i.e., raster cell centers) of the representation. For example, if sensing is performed by moving vehicles on a street network, then all validation points will also be on the street network. Additionally, other observations might also be collected by the same sensor, and the sensor might introduce a particular type of error. Consider the example where sensors move at 15 m/s (≈35 mph), have a sample rate of 5 s, and sample on a street network. This means that for a chosen validation point observation by a sensor, as long as that sensor was operational for the duration of the window, then at least one other sample point will exist within 75 m and 5 s from the validation point. Since sampling by all sensors is constrained to the street network, it's even possible that observations from other sensors fall within this range and may be even closer. Thus, a configuration may score well during validation while in reality in areas outside the street network it's performing poorly.

A robust validation procedure is needed that can assess the accuracy of the representation generated when sensor movement is constrained to movement along a street network.

#### 6.2.1.2 Automatic Parameter Optimization

Once the accuracy of a representation can be assessed in near real-time, then the automatic optimization of parameters to achieve accuracy and runtime requirements can be investigated. In principle, automatic parameter optimization consists of a method to score and rank performance of configurations and a method to explore alternative configurations.

In order to score and rank performance of configurations of the STI-SQO framework, a cost function is needed that calculates a single numeric score or multi-dimensional vector for a configuration based on the performance with respect to different performance metrics. Candidates performance metrics for participation in a cost function include runtime, accuracy (e.g., RMSE), throughput, a measure of the area interpolated (e.g., the number of raster cells interpolated). Additionally, specification of the area a user is interested in could be incorporated into the cost function in order to target interpolation optimization to the regions a user is most interested in. The cost function combines and weights chosen parameters in order to calculate the magnitude of the cost of a configuration used for a window.

Using the cost function, the performance of different configurations of the STI-SQO framework can be ranked and the optimal configuration can be selected. The configurations of the STI-SQO framework are defined by ST-IDW algorithm, $r$, $k$, $p$, $a$, IC, number of sensors, number of cell interpolators, window length, window slide, resolution of representation, and predicate evaluation methods in the case of predicate queries. Additionally, if a raster were to be divided into segments consisting of multiple raster cells, the optimal parameter configuration for a segment (local) may differ from the globally optimal configuration and yield better cost optimization. For example areas a user is not interested in can be pruned which reduces runtime and the best parameters for other areas can be used. The challenge becomes how to do this efficiently as all configuration and segmentations of STI-SQO framework cannot be tested at the same time while still achieving near real-time performance.

In order to be performed in near real-time in a DSE, this approach to parameter optimization would have to be encapsulated into algorithms for stream query operators. The result would be a DSE framework for ST interpolation that can automatically optimize itself to give the best possible results according to a user's parameterized requirements that produces ST snapshots with known accuracy.

### 6.2.2 Predicates over Multiple Fields

This thesis explored efficiently and accurately evaluating predicates over ST fields, specifically for threshold queries on interpolated values for ST fields materialized as rasters. Users are interested in using predicates to analyze multiple fields at the same time in a single query. For instance, a query can be the extraction of a subfield related to the area of a winter storm where wind speed is over 35 mph, snowfall rate is over 0.5 inches per hour, and visibility is less than 1/4 mile. Such a query would take three different fields as input (i.e., wind speed, snowfall rate, and visibility) and output the subfield of each field over the area where the individual predicates are true and then the spatial intersection of these three predicates over the same space and time.

Algorithms are needed to efficiently evaluate such predicates over multiple fields. A naïve approach would evaluate each predicate on its respective field individually and perform a spatial intersection of the result. While this approach would produce valid results, limiting the area over which predicate evaluation needs to be performed can lead to gain in run-time performance. For example, with a conjunction over the predicates $p_1$, $p_2$, and $p_3$ with corresponding result areas of $a_1$,$a_2$,$a_3$ of a larger area $a$, we can restrict the computation as follows: $p_1$ is evaluated over $a$ to produce $a_1$. Then, for the evaluation of $p_2$ the area of predicate evaluation can be restricted to $a_1$ to produce $a_1 \cap a_2$ to be used as input for $p_3$ which produces $a_1 \cap a_2 \cap a_3$ as the final result.

Another option is to rank and order the predicate evaluation based on the most selective predicate being executed first. A predicate with highest *selectivity*, i.e. produces

the smallest area, has the ability to minimize the number of cells over which predicate evaluation is performed. However, this does not guarantee a reduction in runtime. Runtime of interpolation algorithms is impacted by the number of sample points and for local interpolation algorithms the time to compute each cell is impacted by the distribution of sample points. Furthermore, different fields may require different interpolation parameter configurations to produce the most accurate results which also impacts runtime. Thus, while a predicate result region $a_i$ may have the smallest area, it may not have the lowest runtime since it is impacted by the number and distribution of sensors and identifying the optimal predicate ordering is more challenging. Additionally, techniques developed for efficiently evaluating predicates over multiple fields are likely generalizable to support multiple predicates over each field, such as $p_{temperature>28°F} \wedge p_{temperature<35°F}$. Thus, many challenges related to developing algorithms and operators that support multiple queries and representations of ST fields in a stream query operator framework remain as directions for future research.

## Bibliography


[1] Adams, R., Bischof, L.: Seeded Region Growing. IEEE Transactions on Pattern Analysis and Machine Intelligence 16(6), 641–647 (6 1994)

[2] Air Quality Egg: Air Quality Egg, http://airqualityegg.com

[3] Al-Ali, A.R., Zualkernan, I., Aloul, F.: A Mobile GPRS-Sensors Array for Air Pollution Monitoring. IEEE Sensors Journal 10(10), 1666–1671 (10 2010)

[4] ALGLIB: OBSOLETE: Inverse Distance Weighting Interpolation/Fitting (2016), http://www.alglib.net/interpolation/inversedistanceweighting.php

[5] Ali, A.F., Das, S., Vagenas, E.C.: The Generalized Uncertainty Principle and Quantum Gravity Phenomenology. The Twelfth Marcel Grossmann Meeting (3), 2407–2409 (1 2010)

[6] Ali, M.H., Gerea, C., Raman, B.S., Sezgin, B., Tarnavski, T., Verona, T., Wang, P., Zabback, P., Ananthanarayan, A., Kirilov, A., Lu, M., Raizman, A., Krishnan, R., Schindlauer, R., Grabs, T., Bjeletich, S., Chandramouli, B., Goldstein, J., Bhat, S., Li, Y., Di Nicola, V., Wang, X., Maier, D., Grell, S., Nano, O., Santos, I.: Microsoft CEP Server and Online Behavioral Targeting. Proc. VLDB Endow. 2(2), 1558–1561 (8 2009)

[7] Ali, M.H., Aref, W.G., Bose, R., Elmagarmid, A.K., Helal, A., Kamel, I., Mokbel, M.F.: NILE-PDT: A Phenomenon Detection and Tracking Framework for Data Stream Management Systems. In: Proceedings of the 31st international conference on Very large data bases. pp. 1295–1298. VLDB Endowment (2005)

[8] Ali, M.H.: Phenomenon-aware Data Stream Management Systems. Ph.D. thesis, Purdue (2007)

[9] Apache Software Foundation: Apache Flink, https://flink.apache.org

[10] Apache Software Foundation: Apache Samza, https://samza.apache.org

[11] Apache Software Foundation: Apache Spark, https://spark.apache.org

[12] Apache Software Foundation: Storm: Distributed and Fault-Tolerant Realtime Computation (2016), http://storm.apache.org

[13] Arasu, A., Babcock, B., Babu, S., Datar, M., Ito, K., Motwani, R., Nishizawa, I., Srivastava, U., Thomas, D., Varma, R., Widom, J.: STREAM: The Stanford Stream Data Manager. IEEE Data Engineering Bulletin 26(March 2003), 19–26 (2003)

[14] Arasu, A., Babu, S., Widom, J.: The CQL Continuous Query Language: Semantic Foundations and Query Execution. VLDB Journal 15(2), 121–142 (2006)

[15] Arduino: Arduino Products, https://www.arduino.cc/en/Main/Products

[16] Aref, W.G.: PhenomenaBases (position paper) (1997)

[17] Armstrong, M.P.: Temporality in Spatial Databases. Proceedings GIS/LIS 88 (January 1988), 880–889 (1988)

[18] Aurenhammer, F.: Voronoi Diagrams: A Survey of a Fundamental Geometric Data Structure. ACM Computing Surveys. Surv. 23(3), 345–405 (9 1991)

[19] Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and Issues in Data Stream Systems. In: Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. pp. 1–16. PODS '02, ACM, New York, NY, USA (2002)

[20] BeagleBoard.org Foundation: BeagleBoard.org- Community Supported Open Hardware Computers for Making, http://beagleboard.org/

[21] Blue Maestro: Shop Bluetooth Sensor Beacons and Loggers - Blue Maestro, https://www.bluemaestro.com/shop-bluetooth-sensor-beacons-and-loggers/

[22] Bolstad, P.: GIS Fundamentals: A First Text on Geographic Information Systems. Eider Press, fifth edn. (2016)

[23] Bowyer, A.: Computing Dirichlet Tessellations. The Computer Journal 24(2), 162–166 (1 1981)

[24] Broad Group: Air Monitor, http://en.broad.com/ProductShow-3.aspx

[25] Burke, J.A., Estrin, D., Hansen, M., Parker, A., Ramanathan, N., Reddy, S., Srivastava, M.B.: Participatory Sensing. In: WSWâĂŹ06 at SenSys âĂŹ06. Boulder, Colorado, USA (2006)

[26] Camara, G., Egenhofer, M.J., Ferreira, K., Andrade, P., Queiroz, G., Sanchez, A., Jones, J., Vinhas, L.: Fields as a Generic Data Type for Big Spatial Data. In: Geographic Information Science, pp. 159–172. Springer (2014)

[27] Campbell, A.T., Eisenman, S.B., Lane, N.D., Miluzzo, E., Peterson, R.A., Lu, H., Zheng, X., Musolesi, M., Fodor, K., Ahn, G.S.: The Rise of People-Centric Sensing. Internet Computing, IEEE 12(4), 12–21 (7 2008)

[28] Carbone, P., Ewen, S., Haridi, S., Katsifodimos, A., Markl, V., Tzoumas, K.: Apache Flink: Unified Stream and Batch Processing in a Single Engine. Data Engineering 36, 28–38 (2015)

[29] Carney, D., Cetintemel, U., Tatbul, N., Stonebraker, M., Abadi, D.J., Zdonik, S., Convey, C., Lee, S., Cherniack, M.: Aurora: A New Model and Architecture for Data Stream Management. The VLDB Journal The International Journal on Very Large Data Bases 12(2), 120–139 (2003)

[30] Carney, D., Cherniack, M., Convey, C., Lee, S., Seidman, G., Stonebraker, M., Zdonik, S.: Monitoring Streams - A New Class of Data Management Applications. Vldb pp. 215–226 (2002)

[31] Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M.J., Hellerstein, J.M., Hong, W., Krishnamurthy, S., Madden, S., Raman, V., Reiss, F., Shah, M.: TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. CIDR (2003)

[32] Chandrasekaran, S., Franklin, M.J.: Streaming Queries over Streaming Data. Techniques pp. 203–214 (2002)

[33] Chen, J., DeWitt, D.J., Tian, F., Wang, Y.: NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data. pp. 379–390. SIGMOD '00, ACM, New York, NY, USA (2000)

[34] Chino, M., Ishikawa, H., Yamazawa, H.: 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), 145–152 (1993)

[35] Chrisman, N.: Exploring Geographical Information Systems. Wiley, New York, New York, USA, second edn. (5 2001)

[36] Claramunt, C., Thériault, M.: Managing Time in GIS An Event-Oriented Approach. In: Clifford, J., Tuzhilin, A. (eds.) Recent Advances in Temporal Databases, chap. Managing T, pp. 23–42. Springer London, London (1995)

[37] Codd, E.: A Relational Model of Data for Large Shared Data Banks. Communications of the ACM 13(6), 377–387 (1970)

[38] Cova, T.J., Goodchild, M.F.: Extending Geographical Representation to include Fields of Spatial Objects. International Journal of Geographical Information Science 16(6), 509–532 (9 2002)

[39] Elmongui, H.G., Lafayette, W., Aref, W.G.: Challenges in Spatio-temporal Stream Query Optimization. In: MobiDE '06 Proceedings of the 5th ACM International Workshop on Data Engineering for Wireless and Mobile Access. pp. 27–34 (2006)

[40] Erwig, M., Güting, R.H., Schneider, M., Vazirgiannis, M.: Spatio-Temporal Data Types: An Approach to Modeling and Querying Moving Objects in Databases. GeoInformatica 3(3), 269–296 (1999)

[41] ESRI: ArcGIS Release 10 (2012)

[42] Fan, J., Zeng, G., Body, M., Hacid, M.S.: Seeded Region Growing: An Extensive and Comparative Study. Pattern Recognition Letters 26(8), 1139–1156 (2005)

[43] Fan, R., Chakraborty, I., Lynch, N.: Clock Synchronization for Wireless Networks. In: International Conference On Principles Of Distributed Systems. pp. 400–414. Springer (2004)

[44] Farah, C., Schwaner, F., Abedi, A., Worboys, M.: Distributed Homology Algorithm to Detect Topological Events via Wireless Sensor Networks. IET Wireless Sensor Systems 1(3), 151–160 (9 2011)

[45] Faulkner, M., Olson, M., Chandy, R., Krause, J., Chandy, K.M., Krause, A.: The Next Big One: Detecting Earthquakes and other Rare Events from Community-based Sensors. Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks pp. 121–122 (2011)

[46] Fitzner, D., Sester, M.: Estimation of Precipitation Fields from 1-minute Rain Gauge Time Series - Comparison of Spatial and Spatio-temporal Interpolation Methods. International Journal of Geographical Information Science 29(9), 1668–1693 (9 2015)

[47] Forlizzi, L., Güting, R.H., Nardelli, E., Schneider, M.: A Data Model and Data Structures for Moving Objects Databases. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data. pp. 319–330. SIGMOD '00, ACM, New York, NY, USA (2000)

[48] Galić, Z., Baranović, M., Križanović, K., Mešković, E.: Geospatial Data Streams: Formal Framework and Implementation. Data & Knowledge Engineering 91, 1–16 (5 2014)

[49] Galić, Z.: Spatio-Temporal Data Stream Clustering. In: Spatio-Temporal Data Streams, pp. 71–103. Springer New York, New York, NY (2016)

[50] Galić, Z., Mešković, E., Križanović, K., Baranović, M.: OCEANUS: A Spatio-temporal Data Stream System Prototype. In: Proceedings of the Third ACM SIGSPATIAL International Workshop on GeoStreaming. pp. 109–115. IWGS '12, ACM, New York, NY, USA (2012)

[51] Galton, A.: A Formal Theory of Objects and Fields. In: Montello, D.R. (ed.) Spatial Information Theory, chap. A Formal T, pp. 458–473. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)

[52] Galton, A.: Fields and Objects in Space, Time, and Space-time. Spatial Cognition and Computation 1, 39–68 (2004)

[53] Gawlick, D., Mishra, S.: Information Sharing with the Oracle Database. In: Proceedings of the 2Nd International Workshop on Distributed Event-based Systems. pp. 1–6. DEBS '03, ACM, New York, NY, USA (2003)

[54] Gebbert, S., Pebesma, E.: TGRASS: A temporal GIS for field based environmental modeling. Environmental Modelling and Software 53, 1–12 (2014)

[55] Gedik, B., Andrade, H., Wu, K.L., Yu, P.S., Doo, M.: SPADE: The System S Declarative Stream Processing Engine. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data. pp. 1123–1134. SIGMOD '08, ACM, New York, NY, USA (2008)

[56] Gedik, B., Liu, L.: MobiEyes: Distributed Processing of Continuously Moving Queries on Moving Objects in a Mobile System. In: Bertino, E., Christodoulakis, S., Plexousakis, D., Christophides, V., Koubarakis, M., Böhm, K., Ferrari, E. (eds.) Advances in Database Technology - EDBT 2004: 9th International Conference on Extending Database Technology, Heraklion, Crete, Greece, March 14-18, 2004, pp. 67–87. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)

[57] Ghanem, T., Hammad, M., Mokbel, M., Aref, W., Elmagarmid, A.: Incremental Evaluation of Sliding-Window Queries over Data Streams. IEEE Transactions on Knowledge and Data Engineering 19(1), 57–72 (1 2007)

[58] Ghanem, T.M., Aref, W.G., Elmagarmid, A.K.: Exploiting Predicate-window Semantics over Data Streams. SIGMOD Rec. 35(1), 3–8 (3 2006)

[59] Golab, L., Özsu, M.T.: Issues in Data Stream Management. ACM SIGMOD Record 32(2), 5–14 (2003)

[60] Goldman, J., Shilton, K., Burke, J., Estrin, D., Hansen, M., Ramanathan, N., Reddy, S., Samanta, V., Srivastava, M., West, R.: Participatory Sensing: A Citizen-powered Approach to Illuminating the Patterns that Shape our World. Foresight & Governance Project, White Paper pp. 1–15 (2009)

[61] Grenon, P., Smith, B.: SNAP and SPAN: Towards Dynamic Spatial Ontology. Spatial cognition and computation 1(March), 69–103 (2004)

[62] Güting, R.H., Böhlen, M.H., Erwig, M., Jensen, C.S., Lorentzos, N.A., Schneider, M., Vazirgiannis, M.: A Foundation for Representing and Querying Moving Objects. ACM Trans. Database Syst. 25(1), 1–42 (3 2000)

[63] Guttman, A.: R-trees: A Dynamic Index Structure for Spatial Searching. In: Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data. pp. 47–57. SIGMOD '84, ACM, New York, NY, USA (1984)

[64] HabitatMap: AirCasting, `http://aircasting.org`

[65] Hammad, M.A., Aref, W.G., Elmagarmid, A.K.: Stream Window Join: Tracking Moving Objects in Sensor-network Databases. In: Scientific and Statistical Database Management, 2003. 15th International Conference on. pp. 75–84 (7 2003)

[66] Hammad, M.A., Mokbel, M.F., Ali, M.H., Aref, W.G., Catlin, A.C., Elmagarmid, A.K., Eltabakh, M., Elfeky, M.G., Ghanem, T.M., Gwadera, R., Ilyas, I.F., Marzouk, M., Xiong, X.: Nile: A Query Processing Engine for Data Streams (2004)

[67] Heng, I., Zhang, A., Heimbinder, M., Obrok, P.: Development of a Low-cost Mobile Embedded Handheld AirCasting Device. The Technology Interface International Journal 13(1), 14–20 (2012)

[68] Henneböhl, K., Appel, M., Pebesma, E.: Spatial Interpolation in Massively Parallel Computing Environments. In: Geertman, S., Reinhardt, W., Toppen, F. (eds.) 14th AGILE International Conference on Geographic Information Science - Advancing Geoinformation Science for a Changing World. Spring, Utrecht (2011)

[69] Hershberger, J., Shrivastava, N., Suri, S.: Cluster Hull: A Technique for Summarizing Spatial Data Streams. In: 22nd International Conference on Data Engineering (ICDE'06). p. 138 (4 2006)

[70] Huang, F., Liu, D., Tan, X., Wang, J., Chen, Y., He, B.: Explorations of the Implementation of a Parallel IDW Interpolation Algorithm in a Linux Cluster-based Parallel GIS. Computers & Geosciences 37(4), 426–434 (4 2011)

[71] Huang, Y., Zhang, C.: New Data Types and Operations to Support Geo-streams. In: Cova, T.J., Miller, H.J., Beard, K., Frank, A.U., Goodchild, M.F. (eds.) Geographic Information Science: 5th International Conference, GIScience 2008, Park City, UT, USA, September 23-26, 2008. Proceedings, pp. 106–118. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)

[72] Hudnut, K.W., Bock, Y., Galetzka, J.E., Webb, F.H., Young, W.H.: The Southern California Integrated GPS Network (SCIGN). In: The 10th FIG International Symposium on Deformation Measurements. pp. 19–22. Orange California, USA (2001)

[73] Intel: The Intel Galileo Board, `https://software.intel.com/en-us/iot/hardware/galileo`

[74] Jaccard, P.: Étude Comparative de la Distribution Florale dans une portion des Alpes et du Jura. Bulletin de la Société Vaudoise des Sciences Naturelles 37(140), 241–272 (1901)

[75] Jensen, C.S., Clifford, J., Gadia, S.K., Segev, A., Snodgrass, R.T.: A Glossary of Temporal Database Concepts. SIGMOD Rec. 21(3), 35–43 (9 1992)

[76] Kazemitabar, S.J., Banaei-Kashani, F., McLeod, D.: Geostreaming in Cloud. In: Proceedings of the 2nd ACM SIGSPATIAL International Workshop on GeoStreaming. pp. 3–9. IWGS '11, ACM, New York, NY, USA (2011)

[77] Kazemitabar, S.J., Demiryurek, U., Ali, M., Akdogan, A., Shahabi, C.: Geospatial Stream Query Processing Using Microsoft SQL Server StreamInsight. Proc. VLDB Endow. 3(1-2), 1537–1540 (9 2010)

[78] Kemp, K.K.: Fields as a Framework for Integrating GIS and Environmental Process Models. Part 1 : Representing Spatial Continuity. Transactions in GIS 1(3), 219–134 (1997)

[79] Kong, Q., Kwony, Y.W., Schreierz, L., Allen, S., Allen, R., Strauss, J.: Smartphone-based Networks for Earthquake Detection. In: Innovations for Community Services (I4CS), 2015 15th International Conference on. pp. 1–8 (7 2015)

[80] Krige, D.G.: A Satistical Approach to some Mine Valuation and Allied Problems on the Witwatersrand. Ph.D. thesis, University of the Witwatersrand (1951)

[81] Krizanovic, K., Galić, Z., Baranovic, M.: Data Types and Operations for Spatio-temporal Data Streams. In: 2011 IEEE 12th International Conference on Mobile Data Management. pp. 11–14. Ieee, Luleå, Sweden (6 2011)

[82] Kyriakidis, P.C., Journel, A.G.: Geostatistical Space-Time Models: A Review. Mathematical Geology 31(6), 651–684 (1999)

[83] Langran, G.: Time in Geographic Information Systems. Geocarto International 7(2) (6 1992)

[84] Li, J., Heap, A.D.: A Review of Spatial Interpolation Methods for Environmental Scientists. Geoscience Australia (Record 2008/23) (2008)

[85] Li, J., Heap, A.D.: Spatial Interpolation Methods Applied in the Environmental Sciences: A Review. Environmental Modelling & Software 53, 173–189 (2014)

[86] Li, J., Maier, D., Tufte, K., Papadimos, V., Tucker, P.: No Pane, no Gain: Efficient Evaluation of Sliding-window Aggregates over Data Streams. ACM SIGMOD Record 34(1), 39–44 (2005)

[87] Li, L., Revesz, P.: Interpolation Methods for Spatio-temporal Geographic Data. Computers, Environment and Urban Systems 28(3), 201–227 (5 2004)

[88] Li, L., Revesz, P.Z.: A Comparison of Spatio-temporal Interpolation Methods. In: Egenhofer, M., Marks, D. (eds.) GIScience '02: Proceedings of the Second International Conference on Geographic Information Science. pp. 145–160. Springer-Verlag, London, UK (2002)

[89] Li, X., Wang, Y., Li, X., Wang, Y.: Parallelizing Skyline Queries over Uncertain Data Streams with Sliding Window Partitioning and Grid Index. Knowledge and Information Systems 41(2), 277–309 (11 2014)

[90] Liang, Q.: Towards the Continuous Spatio-Temporal Field Model for Sensor Data Streams. Ph.D. thesis, University of Maine (2015)

[91] Liang, Q., Nittel, S., Hahmann, T.: From Data Streams to Fields: Extending Stream Data Models with Field Data Types. In: Ninth International Conference on Geographic Information Science. Springer Lecture Notes in Computer Science, Montreal, Canada (2016)

[92] Liang, Q., Nittel, S., Whittier, J.C., de Bruin, S.: Real-time inverse distance weighting interpolation for streaming sensor data. Transactions in GIS 22(5), 1179–1204 (10 2018)

[93] Libelium: Waspmote, http://www.libelium.com/products/waspmote/

[94] LocationTech: Spatial4j, https://projects.eclipse.org/projects/locationtech.spatial4j

[95] Lorkowski, P., Brinkhoff, T.: Environmental Monitoring of Continuous Phenomena by Sensor Data Streams: A System Approach based on Kriging. Proceedings of EnviroInfo and ICT for Sustainability 2015 (September) (2015)

[96] Lorkowski, P., Brinkhoff, T.: Towards Real-Time Processing of Massive Spatio-temporally Distributed Sensor Data: A Sequential Strategy Based on Kriging. In: Bacao, F., Santos, Y.M., Painho, M. (eds.) AGILE 2015: Geographic Information Science as an Enabler of Smarter Cities and Communities, chap. Towards Re, pp. 145–163. Springer International Publishing, Cham (2015)

[97] Mead, M., Popoola, O., Stewart, G., Landshoff, P., Calleja, M., Hayes, M., Baldovi, J., McLeod, M., Hodgson, T., Dicks, J., Lewis, A., Cohen, J., Baron, R., Saffell, J., Jones, R.: The use of Electrochemical Sensors for Monitoring Urban Air Quality in Low-cost, High-density Networks. Atmospheric Environment 70, 186–203 (5 2013)

[98] Meijster, A., Wilkinson, M.H.F.: A Comparison of Algorithms for Connected Set Openings and Closings. IEEE Transactions on Pattern Analysis and Machine Intelligence 24(4), 484–494 (4 2002)

[99] Mennis, J., Viger, R., Tomlin, C.D.: Cubic Map Algebra Functions for Spatio-Temporal Analysis. Cartography and Geographic Information Science 32(1), 17–32 (1 2005)

[100] Microsoft Corporation: SQL Server–Data Platform, https://www.microsoft.com/en-us/sql-server/

[101] Miller, R.G.: The Jackknife–A review. Biometrika 61(1), 1–15 (4 1974)

[102] Miron Technologies: Dosime: Hybrid Smart Home and Wearable Personal Dosimeter, https://www.mirion.com/products/radiation-detection-and-protection-instruments/personnel-exposure-monitoring-and-access-control/hybrid-smart-home-and-wearable-personal-dosimeter-Dosime/

[103] Mitas, L., Mitasova, H.: Spatial Interpolation. In: Longley, P., Goodchild, M.F., Maguire, D.J., Rhind, D.W. (eds.) Geographical Information Systems: Principles, Techniques, Management and Applications, pp. 481–492. Wiley, 2 edn. (1999)

[104] Mokbel, M.F.: Spatio-Temporal Access Methods: Part 2 (2003-2010) 2, 1–10 (2010)

[105] Mokbel, M.F., Aref, W.G.: PLACE: A Scalable Location-aware Database Server for Spatio-temporal Data Streams. Data Engineering Bulletin 28(3) (2005)

[106] Mokbel, M.F., Aref, W.G.: SOLE: Scalable On-line Execution of Continuous Queries on Spatio-temporal Data Streams. The VLDB Journal 17(5), 971–995 (4 2007)

[107] Mokbel, M.F., Ghanem, T., Aref, W.G.: Spatio-Temporal Access Methods (2003)

[108] Mokbel, M.F., Xiong, X., Aref, W.G.: SINA: Scalable Incremental Processing of Continuous Queries in Spatio-temporal Databases. In: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data. pp. 623–634. Paris, France (2004)

[109] Mokbel, M.F., Xiong, X., Aref, W.G., Hambrusch, S.E., Prabhakar, S., Hammad, M.A.: PLACE: A Query Processor for Handling Real-time Spatio-temporal Data Streams. Proceedings of the International Conference on Very Large Data Bases pp. 1377–1380 (2004)

[110] Mokbel, M.F., Xiong, X., Hammad, M.A., Aref, W.G.: Continuous Query Processing of Spatio-Temporal Data Streams in PLACE. GeoInformatica 9(4), 343–365 (2005)

[111] Moreira, A., Santos, M.Y.: Concave Hull: A k-Nearest Neighbours Approach for The Computation of The Region Occupied By A Set of Points. Proceedings of the 2nd International Conference on Computer Graphics Theory and Applications (GRAPP 2007), Barcelona, Spain pp. 61–68 (2006)

[112] Mosteller, F.: The Jackknife. Revue de l'Institut International de Statistique - Review of the International Statistical Institute 39(3), 363–368 (1971)

[113] Mouratidis, K., Bakiras, S., Papadias, D.: Continuous Monitoring of Top-k Queries over Sliding Windows. Proceedings of the 2006 ACM SIGMOD international conference on Management of data - SIGMOD '06 p. 635 (2006)

[114] Mouratidis, K., Hadjieleftheriou, M., Papadias, D.: Conceptual Partitioning: An Efficient Method for Continuous Nearest Neighbor Monitoring. In: SIGMOD. pp. 634–645 (2005)

[115] Mun, M., Boda, P., Reddy, S., Shilton, K., Yau, N., Burke, J., Estrin, D., Hansen, M., Howard, E., West, R.: PEIR, the Personal Environmental Impact Report, as a Platform for Participatory Sensing Systems Research. In: Proceedings of the 7th international conference on Mobile Systems, Applications, and Services - Mobisys '09. pp. 55–68. ACM Press, New York, New York, USA (2009)

[116] Murty, R.N., Mainland, G., Rose, I., Chowdhury, A.R., Gosain, A., Bers, J., Welsh, M.: CitySense: An Urban-Scale Wireless Sensor Network and Testbed. 2008 IEEE Conference on Technologies for Homeland Security pp. 583–588 (5 2008)

[117] Muthukrishnan, S.: Data streams: Algorithms and Applications. Foundations and Trends in Theoretical Computer Science 1(2), 117–236 (2005)

[118] National Oceanic and Atmospheric Administration: National Data Buoy Center, `http://www.ndbc.noaa.gov/`

[119] Nehme, R.V., Rundensteiner, E.A.: SCUBA: Scalable Cluster-based Algorithm for Evaluating Continuous Spatio-temporal Queries on Moving Objects. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 3896 LNCS, 1001–1019 (2006)

[120] NERACOOS: NERACOOS, http://www.neracoos.org

[121] Nievergelt, J., Hinterberger, H., Sevcik, K.C.: The Grid File: An Adaptable, Symmetric Multikey File Structure. ACM Trans. Database Syst. 9(1), 38–71 (3 1984)

[122] Nittel, S.: Real-time Sensor Data Streams. SIGSPATIAL Special 7(2), 22–28 (9 2015)

[123] Nittel, S., Whittier, J.C., Liang, Q.: Real-time Spatial Interpolation of Continuous Phenomena using Mobile Sensor Data Streams. In: Proceedings of the 20th International Conference on Advances in Geographic Information Systems. pp. 530–533. SIGSPATIAL '12, ACM, New York, NY, USA (2012)

[124] Noghabi, S.A., Paramasivam, K., Pan, Y., Ramesh, N., Bringhurst, J., Gupta, I., Campbell, R.H.: Samza: Stateful Scalable Stream Processing at LinkedIn. Proc. VLDB Endow. 10(12), 1634–1645 (8 2017)

[125] Nyarku, M., Mazaheri, M., Jayaratne, R., Dunbabin, M., Rahman, M.M., Uhde, E., Morawska, L.: Mobile Phones as Monitors of Personal Exposure to Air Pollution: Is this the Future? PLOS ONE 13(2), 1–18 (2018)

[126] Oracle Corporation: Oracle Spatial and Graph 12c (2016), http://www.oracle.com/technetwork/database/options/spatialandgraph/overview/index.html

[127] Patil, D.S., Gavali, A.B., Gavali, S.B.: Review on Indexing Methods in Location Based Services. In: Advance Computing Conference (IACC), 2014 IEEE International. pp. 930–936 (2 2014)

[128] Pebesma, E.J.: Multivariable Geostatistics in S: The Gstat Package. Computers & Geosciences 30(7), 683–691 (8 2004)

[129] Pelekis, N., Theeodoulidis, B., Kopanakis, I., Theeodoulidis, Y.: Literature Review of Spatio-Temporal Database Models. The Knowledge Engineering Review 19(03), 235–274 (2004)

[130] Petit, J.R., Jouzel, J., Raynaud, D., Barkov, N.I., Barnola, J.M., Basile, I., Bender, M., Chappellaz, J., Davis, M., Delaygue, G., Delmotte, M., Kotlyakov, V.M., Legrand, M., Lipenkov, V.Y., Lorius, C., PEpin, L., Ritz, C., Saltzman, E., Stievenard, M.: Climate and Atmospheric History of the Past 420,000 Years from the Vostok Ice Core, Antarctica. Nature 399(6735), 429–436 (6 1999)

[131] Peucker, T.K., Fowler, R.J., Little, J.J., Mark, D.M.: The Triangulated Irregular Network. American Society of Photogrammetry Proceedings of Digital Terrain Models Symposium pp. 96–103 (1978)

[132] Peuquet, D.J., Duan, N.: An Event-based Spatiotemporal Data Model (ESTDM) for Temporal Analysis of Geographical Data. International Journal of Geographical Information Systems 9(1), 7–24 (1995)

[133] PostGIS: PostGIS (2016)

[134] Preparata, F.P., Hong, S.J.: Convex Hulls of Finite Sets of Points in Two and Three Dimensions. Commun. ACM 20(2), 87–93 (2 1977)

[135] R Core Team: R: A Language and Environment for Statistical Computing (2015), http://www.r-project.org/

[136] Raper, J., Livingstone, D.: Development of a Geomorphological Spatial Model Using Object-oriented Design. International Journal of Geographical Information Systems 9(4), 359–383 (7 1995)

[137] Raspberry Pi Foundation: Raspberry Pi - Teach, Learn, and Make with Raspberry Pi, https://www.raspberrypi.org/

[138] Renka, R.J.: Multivariate Interpolation of Large Sets of Scattered Data. ACM Trans. Math. Softw. 14(2), 139–148 (6 1988)

[139] Roemmich, D.H., Davis, R.E., Riser, S.C., Owens, W.B., Molinari, R.L., Garzoli, S.L., Johnson, G.C.: The Argo Project Global Ocean Observations for Understanding and Prediction of Climate Variability. Tech. rep., DTIC Document (2003)

[140] Rundensteiner, E.A., Ding, L., Sutherland, T., Zhu, Y., Pielech, B., Mehta, N.: CAPE: Continuous Query Engine with Heterogeneous-grained Adaptivity. In: Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30. pp. 1353–1356. VLDB '04, VLDB Endowment (2004)

[141] Rundensteiner, E.A., Ding, L., Zhu, Y., Sutherland, T., Pielech, B.: CAPE: A Constraint-Aware Adaptive Stream Processing Engine. In: Chaudhry, N.A., Shaw, K., Abdelguerfi, M. (eds.) Stream Data Management, pp. 83–111. Springer US, Boston, MA (2005)

[142] Safecast: Safecast (2016), safecast.org

[143] Samsung: Samsung Galaxy S4, http://www.samsung.com/us/mobile/phones/galaxy-s/samsung-galaxy-s4-generic-black-mist-sch-i545zkalra/

[144] Sanchez, L., Galache, J.A., Gutierrez, V., Hernandez, J.M., Bernat, J., Gluhak, A., Garcia, T.: SmartSantander: The Meeting Point between Future Internet Research and Experimentation and the Smart Cities. In: Future Network Mobile Summit (FutureNetw), 2011. pp. 1–8 (6 2011)

[145] Saska, M., Langr, J., P\vreučil, L.: Plume Tracking by a Self-stabilized Group of Micro Aerial Vehicles. In: Hodicky, J. (ed.) Modelling and Simulation for Autonomous Systems: First International Workshop, MESAS 2014, Rome, Italy, May

5-6, 2014, Revised Selected Papers, pp. 44–55. Springer International Publishing, Cham (2014)

[146] Schenato, L., Gamba, G.: A Distributed Consensus Protocol for Clock Synchronization in Wireless Sensor Network. In: Decision and Control, 2007 46th IEEE Conference on. pp. 2289–2294 (12 2007)

[147] SHARP: PANTONE5 SoftBank 107SH (2012), http://www.sharp.co.jp/products/sb107sh/

[148] Shekhar, S., Evans, M.R., Kang, J.M., Mohan, P.: Identifying Patterns in Spatial Information: A Survey of Methods. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 1(3), 193–214 (2011)

[149] Shepard, D.: A Two-dimensional Interpolation Function for Irregularly-spaced Data. In: Proceedings of the 1968 23rd ACM National Conference. pp. 517–524. ACM '68, ACM, New York, NY, USA (1968)

[150] Shih, F.Y., Cheng, S.: Automatic Seeded Region Growing for Color Image Segmentation. Image and Vision Computing 23(10), 877–886 (2005)

[151] Sinnott, R.W.: Virtues of the Haversine. Sky Telescope 68, 158 (12 1984)

[152] Snodgrass, R., Ahn, I.: A Taxonomy of Time Databases. In: Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data. pp. 236–246. SIGMOD '85, ACM, New York, NY, USA (1985)

[153] Snodgrass, R.T.: Temporal Databases. In: Frank, A.U., Campari, I., Formentini, U. (eds.) Theories and Methods of Spatio-Temporal Reasoning in Geographic Space, chap. Temporal d, pp. 22–64. Springer Berlin Heidelberg, Berlin, Heidelberg (1992)

[154] Srinivasan, B.V., Duraiswami, R., Murtugudde, R.: Efficient Kriging for Real-time Spatio-temporal Interpolation. In: Proceedings of the 20th Conference on Probability and Statistics in the Atmospheric Sciences. pp. 228–235. American Meteorological Society Atlanta GA (2010)

[155] Stonebraker, M., Cetintemel, U.: "One Size Fits All": An Idea Whose Time Has Come and Gone. In: 21st International Conference on Data Engineering (ICDE'05). pp. 2–11 (4 2005)

[156] Stonebraker, M., Çetintemel, U., Zdonik, S.: The 8 Requirements of Real-Time Stream Processing. ACM SIGMOD Record 34(4), 42–47 (12 2005)

[157] Tatbul, N., Zdonik, S.: Load Shedding in a Data Stream Manager. on Very large data (2003)

[158] Terry, D., Goldberg, D., Nichols, D., Oki, B.: Continuous Queries over Append-Only Databases. In: Proceedings of the 1992 ACM SIGMOD international conference on Management of data - SIGMOD '92. pp. 321–330. SIGMOD '92, ACM Press, New York, New York, USA (1992)

[159] Tobler, W.R.: A Computer Movie Simulating Urban Growth in the Detroit Region. Economic Geography 46, 234–240 (1970)

[160] Tomczak, M.: Spatial Interpolation and its Uncertainty Using Automated Anisotropic Inverse Distance Weighting (IDW) - Cross-Validation/Jackknife Approach. Journal of Geographic Information and Decision ... 2(2), 18–30 (1998)

[161] Variable Inc: NODE+ iOS and Android Wireless Sensor Platform (2015), http://docs.wixstatic.com/ugd/54926a_d74fdd41d95d41218dfa5900e727ca4e.pdf

[162] Venkatesan, B.: Feasibility Study of Continuous Real-Time Spatial Interpolation of Phenomena using Built-In Functionlity of a Commercial Data Stream Management System. Tech. rep., The University of Maine (2013)

[163] Vieira, M.R., Bakalov, P., Tsotras, V.J.: On-line Discovery of Flock Patterns in Spatio-temporal Data. In: Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. pp. 286–295. GIS '09, ACM, New York, NY, USA (2009)

[164] Viglas, S.D., Naughton, J.F., Burger, J.: Maximizing the Output Rate of Multi-way Join Queries over Streaming Information Sources. In: Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29. pp. 285–296. VLDB '03, VLDB Endowment (2003)

[165] Whittier, J., Nittel, S., Liang, Q., Plummer, M.: Towards Window Stream Queries over Continuous Phenomena. In: Conf Proc of 4th International Workshop on "Geostreaming", in conjunction with SIGSPATIAL. pp. 1–10. Orlando, FL (2013)

[166] Wilensky, U.: NetLogo: Center for Connected Learning and Computer-based Modeling (1999), http://ccl.northwestern.edu/netlogo/

[167] Wimoto Technologies Inc.: Wimotos - Tiny Wireless Helpers For Your Life (2013), https://www.indiegogo.com/projects/wimotos-tiny-wireless-helpers-for-your-life--47#/

[168] Wolfson, O., Xu, B., Chamberlain, S., Jiang, L.: Moving Objects Databases: Issues and Solutions. In: Rafanelli, M., Jarke, M. (eds.) Proceedings of the 10th International Conference on Scientific and Statistical Database Management. pp. 111–122. IEEE Computer Society (1998)

[169] Worboys, M., Duckham, M.: GIS: A Computing Perspective. CRC Press, Inc. (2004)

[170] Worboys, M.F.: A Unified Model for Spatial and Temporal Information. The Computer Journal 37(1), 26–34 (1 1994)

[171] Worboys, M.F.: A Generic Model for Spatio-bitemporal Geographic Information. Spatial and temporal reasoning in geographic information systems pp. 25–39 (1998)

[172] Wotawa, G., Skomorowski, P.: Long-range Transport of Particulate Radionuclides from the Fukushima NPP Accident: Sensitivity Analysis for Wet Deposition. In: EGU General Assembly 2012. p. 10494. Vienna, Austria (2012)

[173] Yu, J.J.Q., Li, V.O.K., Lam, A.Y.S.: Sensor Deployment for Air Pollution Monitoring Using Public Transportation System. In: 2012 IEEE Congress on Evolutionary Computation. pp. 1–7 (6 2012)

[174] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M.J., Shenker, S., Stoica, I.: Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation. p. 2. NSDI'12, USENIX Association, Berkeley, CA, USA (2012)

[175] Zhang, C., Huang, Y.: Querying Streaming Point Clusters as Regions. In: Proceedings of the ACM SIGSPATIAL International Workshop on GeoStreaming. pp. 43–50. ACM (2010)

[176] Zhong, X., Kealy, A., Duckham, M.: Stream Kriging: Incremental and Recursive Ordinary Kriging over Spatiotemporal Data Streams. Computers & Geosciences 90, 134–143 (2016)

[177] Zhong, X., Kealy, A., Sharon, G., Duckham, M.: Spatial Interpolation of Streaming Geosensor Network Data in the RISER System. In: W2GIS 2015. vol. LNCS 9080, pp. 161–177. Springer Lecture Notes in Computer Science 9080 (2015)

# APPENDIX A

# ACRONYMS

**ADT** Abstract Data Type.

**ANN** Aggregate Nearest Neighbor.

**BF** Breadth-First.

**CPU** Central Processing Unit.

**CQL** Continuous Query Language.

**CST** Cylindrical Shell Template.

**DAG** Directed, Acyclic Graph.

**DBMS** Database Management System.

**DSE** Data Stream Engine.

**ESTDM** Event oriented Spatio-Temporal Data Model.

**GIS** Geographic Information System.

**GPS** Global Positioning System.

**GPU** Graphics Processing Unit.

**IC** Interpolation Center.

**IDW** Inverse Distance Weighting.

**ITC** Isotropic Time Cell.

**K** $K = 1024$.

**kNN** k-Nearest Neighbors.

**NN** Nearest Neighbors.

**NRMSE** Normalized Root Mean Square Error.

**NST** Nested Shell Template.

**RDBMS** Relational Database Management System.

**RMSE** Root Mean Square Error.

**SDBMS** Spatial Database Management System.

**SL** Scan Line.

**SQL** Structured Query Language.

**ST** Spatio-Temporal.

**ST a$k$-Shell** ST Adaptive $k$-Shell.

**ST field** Spatio-Temporal Field.

**ST grid index** Sliding Spatio-Temporal Grid Index.

**ST Shell** ST Shell Approach.

**ST-DBMS** Spatio-Temporal Database Management System.

**ST-IDW** Spatio-Temporal Inverse Distance Weighting.

**STI-SQO framework** ST Interpolation Stream Query Operator Framework.

**STP-SQO framework** ST Predicate Stream Query Operator Framework.

**TIN** Triangulated Irregular Network.

**VOC** Volatile Organic Compounds.

# APPENDIX B

# PSEUDOCODE FOR STI-SQO FRAMEWORK

**Algorithm 1** Index Loader

```
1: inputTuples ← a queue of ST sensor observation tuples from the InputAdapter
2: inputTimeBlocks ← a queue of empty time blocks from IndexRecycler
3: outputTimeBlocks ← a queue filled time blocks to IndexManager
4: currentTimeBlock ← inputTimeBlocks.dequeue();
5: while true do
6:     tuple ← inputTuples.dequeue()
7:     while true do
8:         if Timestamp of tuple is in currentTimeBlock then
9:             Insert tuple in ST grid index contained in currentTimeBlock
10:            break
11:        else
12:                ▷ Output the current time block and set currentTimeBlock equal to the next time block
13:            outputTimeBlocks.enqueue(currentTimeBlock)
14:            currentTimeBlock ← inputTimeBlocks.dequeue()
```

**Algorithm 2** Index Manager

```
1: inputTimeBlocks ← a queue of filled time blocks from IndexLoader
2: outputTimeBlocks ← a queue of expired time blocks to the IndexRecycler
3: recyclerOutput ← a queue of time blocks cleared by the IndexRecycler
4: Initialize stGridIndex as 1D array of subblocks
5: Initialize recyclerOutput with desired number of TimeBlocks
6: while true do
7:     nextTimeBlock ← inputTimeBlocks.dequeue()
8:     Wait for InterpolationManager to finish
9:     if stGridIndex contains a full window then
10:        expiredTimeBlock ← oldest time block in stGridIndex
11:        outputTimeBlocks.enqueue(expiredTimeBlock)
12:    Add nextTimeBlock to stGridIndex
13:    if stGridIndex contains a full window then
14:        Tell InterpolationManager to begin interpolation for the next window
```

**Algorithm 3** Interpolation Manager

1: $numCellInterpolators \leftarrow$ the number of cell interpolators
2: $outputCellsToInterpolators$
3: $windowStart \leftarrow$ start of current window from IndexManager
4: **while** true **do**
5: Wait for all GridCellInterpolators to finish
6: Tell IndexManager that interpolation is finished
7: Wait for GridIndex to have all time blocks in window
8: **for all** GridCells with time == InterpolationCenter + windowStart **do**
9: Enqueue each grid cell in $outputCellsToInterpolators$
10: **for** $n$ in $[0, numCellInterpolators)$ **do**
11: Enqueue $windowEndMarker$ in $outputCellsToInterpolators$
12: Start all GridCellInterpolators

**Algorithm 4** Grid Cell Interpolator (Abstract Class)

```
1: interpolationCenter
2: anisotropyRatio
3: inputCellsToInterpolate
4: outputInterpolatedCells
5: stGrid ← stGridIndex
6: while true do
7:     Wait for start message from InterpolationManager
8:     while true do
9:         cellToInterpolate ← inputCellsToInterpolate.dequeue()
10:        if cellToInterpolate is not windowEndMarker then
11:            InterpolatedGridCell ← interpolateGridCell(cellToInterpolate)
12:            outputInterpolatedCells.enqueue(InterpolatedGridCell)
13:        else
14:            break
15:    Tell InterpolationManager finished interpolating window
16: function stDistanceBetween(gridCell, tuple, snapshotTime)
17:    spatialDist ←
18:      haversineDistInDeg(gridCell.lat(), gridCell.lon(), tuple.x(), tuple.y())
19:    studyDiag ← diagonal of study area in degrees
20:    gridDiag ← diagonal of output grid in grid cells
21:    temporalDist ← 1/anisotropyRatio
22:    temporalDist ← temporalDistance * (studyDiag/gridDiag)
23:    temporalDist ← temporalDistance * (gridCell.time − tuple.time)
24:    return sqrt(spatialDist * spatialDist + temporalDist * temporalDist)
25: function updateCalc(gridCell_i, tupleList)
26:    for all tuple in tupleList do
27:        tupleCount ← tupleCount + 1
28:        distance ← stDistanceBetween(gridCell_i, tuple, snapshotTime)
29:        weightedDistance ← distance ^ p
30:        numerator ← numerator + weightedDistance × tuple.getValue()
31:        denominator ← denominator + weightedDistance
```

**Algorithm 5** Subclass of Grid Cell Interpolator: ST Shell

```
1: function InterpolateGridCell(gridCell)
2:     x ← gridCell.getX()
3:     y ← gridCell.getX()
4:     numerator ← 0
5:     denominator ← 0
6:     tupleCount ← 0
7:     u ← noDataCode
8:     stGrid ← stGridIndex
9:     for all cell c in CST do
10:        updateCalc(gridCell, stGrid.getTuplesInWindow(x + c.x, y + c.y))
11:        updateCalc(gridCell, stGrid.getTuplesInWindow(x − c.x, y + c.y))
12:        updateCalc(gridCell, stGrid.getTuplesInWindow(x − c.x, y − c.y))
13:        updateCalc(gridCell, stGrid.getTuplesInWindow(x + c.x, y − c.y))
14:    if tupleCount > 0 and denominator! = 0 then
15:        u ← numerator/denominator
16:    gridCell.setValue(u)
17:    return gridCell
```

**Algorithm 6** Subclass of Grid Cell Interpolator: ST a$k$-Shell

```
1: function InterpolateGridCell(gridCell)
2:     x ← gridCell.getX()
3:     y ← gridCell.getX()
4:     t ← snapshotTime
5:     numerator ← 0
6:     denominator ← 0
7:     tupleCount ← 0
8:     u ← noDataCode
9:     for all Shell s in NST do
10:        for all Cell c in s.list do
11:            for all Eight +/- sign combinations c.x, c.y, and c.t do
12:                x2 ← ±c.x + x
13:                y2 ← ±c.y + y
14:                t2 ← ±c.t + t
15:                tuples ←stGrid.getTuplesInIsoTimeCell(x2, y2, t2)
16:                updateCalc(gridCell, tuples)
17:            if tupleCount >= k then
18:                break
19:    if tupleCount > 0 and denominator! = 0 then
20:        u ← numerator/denominator
21:    gridCell.setValue(u)
22:    return gridCell
```

# APPENDIX C

# PSEUDOCODE FOR STP-SQO FRAMEWORK

**Algorithm 7** Naïve

```
1: outputInterpolatedCells ← stream of all interpolated raster cells
2: output ← stream output queue for filtered cells
3: while true do
4:     cellToTest ← outputInterpolatedCells.dequeue()
5:     if cellToTest is windowEndMarker or cellToTest satisfies predicate then
6:         output.put(cellToTest)
```

**Algorithm 8** Tile-based

```
1: windowTuples ← tuples in window
2: predicate ← value predicate
3: seedTuples ← filter(windowTuples, predicate)
4: seedCells ← set of cells having seeds
5: seedTiles ← set of tiles having seed cells
6: cellsToInterpolate ← set of all cells contained within seedTiles
7: output ← stream output queue for filtered cells
8:
9: for all cell in CellsToInterpolate do
10:     InterpolatedGridCell ← interpolateGridCell(cellToInterpolate)
11:     if InterpolatedGridCell satisfies predicate then
12:         output.put(InterpolatedGridCell)
```

**Algorithm 9** Phenomenon-Aware

```
1: windowTuples ← tuples in window
2: predicate ← value predicate
3: interpolatedCells ← interpolated cells from previous window
4: seedTuples ← filter(windowTuples, predicate)
5: seedCells ← set of cells having seeds
6: seedTiles ← set of tiles having seed cells
7: interiorTiles ←
8: exteriorTiles ←
9: boundaryTiles ←
10: noDataTiles ←
11: tilesToNotExpand ← (interiorTiles ∪ boundaryTiles) ∩ seedTiles
12: tilesToExpand ← (ExteriorTiles ∪ noDataTiles) ∩ seedTiles
13: expandedTiles ← expand(tilesToExpand)
14: tilesToInterpolate ← tilesToNotExpand ∪ expandedTiles
15: cellsToInterpolate ← set of all cells in tilesToInterpolate
16:
17: for all cell in CellsToInterpolate do
18:     InterpolatedGridCell ← interpolateGridCell(cellToInterpolate)
19:     if InterpolatedGridCell satisfies predicate then
20:         output.put(InterpolatedGridCell)
```

**Algorithm 10** Region Growing (Abstract Class)

```
1: width ← width of output grid
2: height ← height of output grid
3: windowTuples ← tuples in window
4: output ← stream output queue
5: predicate ← value predicate
6: seedTuples ← filter(windowTuples, predicate)
7: seedCells ← set of cells having seeds
8: queueMask  array(width * height) ← false
9:
10: for all seedCell in seedCells do
11:     growRegion(seedCell, output)
12:
13: function GROWREGION(seedCell, output)
14: function HASALREADYBEENQUEUED(gridCell)
15:     index ← gridCell.x * width + gridCell.y
16:     return queueMask[index]
17: function SETALREADYBEENQUEUED(gridCell)
18:     index ← gridCell.x * width + gridCell.y
19:     queueMask[index] ← true
```

**Algorithm 11** Subclass of Region Growing: Breadth-First Region Growing

```
1: function growRegion(seedCell, output)
2:     if hasAlreadyBeenQueued(seedCell) then
3:         return
4:     setHasAlreadyBeenQueued(seedCell)
5:     regionQueue ← new Queue
6:     regionQueue.enqueue(seedCell)
7:     while regionQueue is not empty do
8:         cellToInterpolate ← regionQueue.dequeue()
9:         value ← interpolateGridCell(cellToInterpolate)
10:        if value satisfies predicate then
11:            output.put(cell and value)
12:            neighboringCells ← NSEW neighbors of cellToInterpolate
13:            for all neighbor in neighboringCells do
14:                if ¬ hasAlreadyBeenQueued(neighbor) then
15:                    setHasAlreadyBeenQueued(neighbor)
16:                    regionQueue.enqueue(neighbor)
```

**Algorithm 12** Subclass of Region Growing: ScanLine Region Growing

```
1: function GROWREGION(seedCell, output)
2:     if hasAlreadyBeenQueued(seedCell) then
3:         return
4:     setHasAlreadyBeenQueued(seedCell)
5:     regionQueue ← new Queue, spanQueue ← new Queue, isSeed ← true
6:     spanQueue.enqueue(generateSpan(seedCell.y,seedCell.x,seedCell.x))
7:     while spanQueue is not empty do
8:         span ← spanQueue.dequeue()
9:         leftIndex ← NULL, rightIndex ← NULL
10:        cellsInSpan ← all cells in span from span.l to span.r
11:        regionQueue.enqueue(cellsInSpan)
12:        while regionQueue is not empty do                    ▷ Move right
13:            cellToInterpolate ← regionQueue.dequeue()
14:            value ← interpolateGridCell(cellToInterpolate)
15:            if value satisfies predicate then
16:                output.put(cell and value)
17:                rightIndex ← cellToInterpolate.x
18:                if leftIndex is NULL then leftIndex ← cellToInterpolate.x
19:                nextCell ← right neighbor of cellToInterpolate
20:                if ¬ hasAlreadyBeenQueued(nextCell) then
21:                    regionQueue.enqueue(nextCell)
22:                    setHasAlreadyBeenQueued(nextCell)
23:            else
24:                if isSeed then return
25:            isSeed ← false
26:        nextCell ← left neighbor of span.l                   ▷ Prepare to Move left
27:        if ¬ hasAlreadyBeenQueued(nextCell) then
28:            regionQueue.enqueue(nextCell)
29:        while regionQueue is not empty do                    ▷ Move left
30:            cellToInterpolate ← regionQueue.dequeue()
31:            value ← interpolateGridCell(cellToInterpolate)
32:            if value satisfies predicate then
33:                output.put(cell and value)
34:                leftIndex ← cellToInterpolate.x
35:                if rightIndex is NULL then rightIndex ← cellToInterpolate.x
36:                nextCell ← left neighbor of cellToInterpolate
37:                if ¬ hasAlreadyBeenQueued(neighbor) then
38:                    regionQueue.enqueue(nextCell)
39:                    setHasAlreadyBeenQueued(nextCell)
40:        if leftIndex is not NULL then
41:            spanQueue.enqueue(generateSpan(y + 1, leftIndex, rightIndex + 1))
42:            spanQueue.enqueue(generateSpan(y − 1, leftIndex, rightIndex + 1))
```

**Algorithm 13** Scan Line Span

```
1: function GENERATESPAN(y,leftIndex,rightIndex)
2:     spanStarted ← false
3:     spanLeft, spanRight
4:     for i from leftIndex to rightIndex do
5:         if ¬ hasAlreadyBeenQueued(gridCell(i,y)) then
6:             if ¬ spanStarted then
7:                 spanStarted ← true
8:                 spanLeft ← i
9:             spanRight ← i
10:            setHasAlreadyBeenQueued(gridCell(i,y))
       return span(y, spanLeft , spanRight)
```

## BIOGRAPHY OF THE AUTHOR

John Chandler Whittier was born in Camden, ME and graduated from high school in Rockport, ME. He graduated from The University of Maine in 2010 with a Bachelor of Science in Information Systems Engineering. During his graduate study at The University of Maine, he was interested in geo-sensor data stream processing and published papers related to his study listed below.

- Liang, Qi., Nittel, S., Whittier, J.C. and de Bruin, S. 2018. Real-time Inverse Distance Weighting Interpolation for Streaming Sensor Data. Transactions in GIS 22(5), 1179–1204.
- Whittier, J.C., Nittel, S. and Subasinghe, I. 2017. Real-time Earthquake Monitoring with Spatio-Temporal Fields. International Symposium on Spatiotemporal Computing (Cambridge, MA, 2017).
- Whittier, J.C., Liang, Q. and Nittel, S. 2014. Evaluating Stream Predicates over Dynamic Fields. Proceedings of the 5th ACM SIGSPATIAL International Workshop on GeoStreaming (Dallas, TX, 2014), 2-11.
- Whittier, J.C., Nittel, S., Liang, Q. and Plummer, M.A. 2013. Towards Window Stream Queries over Continuous Phenomena. Proceedings of the 4th ACM SIGSPATIAL International Workshop on GeoStreaming (Orlando, FL, 2013), 1-10.
- Nittel, S., Whittier, J.C. and Liang, Q. 2012. Real-time spatial interpolation of continuous phenomena using mobile sensor data streams. Proceedings of the 20th International Conference on Advances in Geographic Information Systems (New York, NY, USA, 2012), 530-533.
- Nittel, S., Dorr, C. and Whittier, J.C. 2012. LocalAlert: Simulating Decentralized Ad-Hoc Collaboration in Emergency Situations. Geographic Information Science SE -

11. N. Xiao, M.-P. Kwan, M. Goodchild, and S. Shekhar, eds. Springer Berlin Heidelberg. 1460159.

He is a candidate for the Doctor of Philosophy degree in Spatial Information Science and Engineering from The University of Maine in August 2018.