Journal of Ubiquitous Systems & Pervasive Networks
Volume 15, No. 2 (2021) pp. 33-41

Unbounded Spatial Data Stream Query Processing using Spatial Semijoins

Wendy Osborn

Department of Mathematics and Computer Science, University of Lethbridge, Lethbridge, Alberta, Canada, T1K 3M4

Abstract

In this paper, the problem of query processing in spatial data streams is explored, with a focus on the spatial join operation. Although the spatial join has been utilized in many proposed centralized and distributed query processing strategies, for its application to spatial data streams the spatial join operation has received very little attention. One identified limitation with existing strategies is that a bounded region of space (i.e., spatial extent) from which the spatial objects are generated needs to be known in advance. However, this information may not be available. Therefore, two strategies for spatial data stream join processing are proposed where the spatial extent of the spatial object stream is not required to be known in advance. Both strategies estimate the **common region** that is shared by two or more spatial data streams in order to process the spatial join. An evaluation of both strategies includes a comparison with a recently proposed approach in which the spatial extent of the data set is known. Experimental results show that one of the strategies performs very well at estimating the common region of space using only incoming objects on the spatial data streams. Other limitations of this work are also identified.

Keywords: *spatial data streams, stream query processing, spatial join, performance*

1. Introduction

Many applications that continuously generate data exist today [1]. One example is a stock market that generates data on the volume of stocks that are traded every day. As shares are traded, this information can be transmitted on a data stream. Another example is a remote sensing application where alterations to specific locations of land are tracked. Any identified differences to a location can be indicated by using a region (e.g., rectangle) that identifies the location in space of the specific location. The latter is an example of a spatial data stream in which object data (i.e., objects that represent specific locations in space) [3], as opposed to simple numeric or alphanumeric data, are generated and transmitted along the data stream.

The data in two or more data streams can be related based on specific criteria. For example, given two spatial data streams – one stream which tracks regions of a large commercial farm where the yield of a particular area falls below some expected threshold, and a second stream that tracks regions where a high number of pests (e.g., rats, mice) are detected. If areas between these two streams overlap – partially or completely – this may provide one indicator of the reason behind the unexpectedly lower yield. This is an example of a spatial join, which relates objects between two sets using a spatial predicate (e.g., overlap).

Over a significant amount of time, streaming data applications generate a continuous and very large volume of data [1],[2]. This leads to several issues. First, in addition to its (eventual) very large size, not all of the data that is required for processing has arrived at the time it is needed. In addition, in a data stream application the amount of memory available for maintaining data is very small when compared to the amount of incoming data, which means some data items must be deleted in order to make room for the incoming items. Finally, many data items in the stream can become invalid over time – for example, when a change to a specific data item occurs and the update has not yet arrived [1],[2] – and must be deleted. These issues are magnified when dealing with spatial data. Since its size is greater than that of alphanumeric data, fewer instances of it will fit in the same amount of streaming data memory as alphanumeric data. In addition, the reasons for spatial data becoming invalid may also differ, as we are dealing with data with multiple dimensions and in specific locations in space. Referring back to the earlier example, if the number of pests in an of farmland is no longer considered high, this would render the region representing it is considered no longer valid.

These issues with spatial data streams also introduce challenges with strategies that operate on the data. One operation that needs to be reconsidered in in the context of spatial data streams is the spatial join [4]. Given two object sets $X=\{x_1,$

* Corresponding author. Tel.: 1(403) 329-2294
Fax: +9876543210; E-mail: wendy.osborn@uleth.ca

DOI: 10.5383/JUSPN.03.02.005

$x_2 \ldots x_m\}$ and $Y=\{y_1, y_2 \ldots y_n\}$, a spatial join matches pairs of objects between the sets (i.e., x_i, y_i) using a spatial predicate, such as overlap, containment, nearest neighbour, or direction. However, traditional spatial join processing requires that all objects are available at the time of the join. However, this may not be possible for objects that are transmitted continuously on a spatial data stream. In this situation, each object must be processed for the spatial join immediately on arrival, as it may have already been required for the join. In addition, decisions must be made on: 1) whether to keep this object, as it may be required for future joining with an object that has not yet arrived, 2) whether to delete the object later on, if it has been found to be invalid, and 3) whether to delete one or more objects in order to make room for new ones that have arrived [1].

Therefore, new strategies are needed for managing spatial stream data, as well as for processing operations in light of the data stream environment. This work focuses on the latter, and specifically on the adaptation of the spatial join operation for processing spatial data streams.

Previously proposed strategies that employ spatial joins for query processing in spatial databases can be classified as follows: for centralized queries [5],[6],[7],[8],[9],[10], for distributed queries [11],[12],[13],[14],[15],[16], and for spatial data stream queries [3],[17].

Significant work has been proposed for both centralized and distributed query processing. However, we see that limited work has been proposed for spatial data stream query processing involving spatial joins. In addition, the limitations of these two approaches include the repeated spatial join processing of the same objects, and the requirement that the containment area (i.e., spatial extent) of all objects generated by a spatial data stream must be known in advance. It is not always possible to know this latter requirement – at least not accurately.

Therefore, to attempt to overcome the limitations of the previously strategies, two new strategies are proposed for spatial data stream joining strategies. Both strategies do not require that the spatial extent from which the spatial data stream objects are drawn be known in advance, and do not perform repeated spatial joining of the same two objects. Both strategies estimate a common spatial extent for all spatial data streams participating in the spatial join, from the incoming objects themselves. The first strategy (Sliding Window) uses an initial sliding window of objects obtained from two spatial data streams to estimate a common spatial extent. The second strategy (Incremental) utilizes the first object from both incoming spatial data streams to create an initial common spatial extent, and incrementally (and selectively) grows the common region using more objects from the spatial data streams as they arrive. The experimental evaluation that was performed includes a comparison with an existing approach that uses a pre-determined common spatial extent. The results show that the Sliding Window strategy significantly outperforms the Incremental strategy. It also obtains results that are comparable to the existing approach.

The rest of this paper proceeds as follows. Section 2 summarizes related work in different areas of spatial join processing. Section 3 presents the background information that is needed for both proposed strategies. Section 4 presents the two proposed strategies that attempt to overcome the limitation of existing strategies. Section 5 presents the methodology and results of the performance evaluation and comparison. Finally, Section 6 concludes the paper and presents some future research directions that arise from this work.

2. Related Work

In this section a summary of previously proposed strategies that utilize a spatial join for query processing is presented. Spatial query processing strategies that utilize the spatial join can be organized as follows:

- for centralized queries [5],[6],[7],[8],[9],[10],
- for distributed queries [11],[12],[13],[14],[15],[16], and
- for queries on spatial data streams [3],[17].

Several strategies for processing spatial joins focus on partitioning approaches. Patel and Dewitt [5] propose a partition-based strategy for performing a spatial merge join. Using the spatial extent for all objects participating in the join, the authors partition the space into multiple regions before creating one bucket for each region that contains the objects for that region. Each bucket is the processed for spatial predicate matches between objects. Zhou *et al.* [7] extend the approach of Patel and Dewitt for a parallel execution environment, considering both CPU and communication costs. Arge *et al.* [8] also propose a partition-based spatial join strategy that also utilizes plane sweeping. The spatial extent is partitioned vertically, with each strip processed separately using a plane sweeping algorithm.

Strategies for processing distributed spatial queries focus on improving the spatial semijoin approach for handling spatial joins and applying the new strategy to distributed query processing. Abel *et al.* [11] and Tan *et al.* [12] propose two strategies to speed up the processing of a semijoin-based spatial join. The first utilized the leaf-level minimum bounding rectangles obtained from a spatial index by an R-tree to represent the spatial attribute, while the second utilized a quad-tree key representation of the attribute. The authors recommend the first for a larger spatial attribute and the second for a smaller one. Karam and Petry [14] extend the first approach of [11],[12] by considering minimum bounding rectangles from different levels of an R-tree for the spatial attribute. Kalnis et al. propose a spatial join strategy where the query site is a mobile device. Their goal is to reduce the data transmission cost from two non-cooperating sites by pruning data that will not be part of the final result before it is transmitted to the mobile device. Farruque and Osborn [15] also propose two strategies to improve the processing and data transmission cost of a semijoin-based spatial join. Both utilize a partition of the common spatial extent that objects occupy, and also a representation of the cells that contain objects. Their first strategy uses the indexes of the partition (i.e., where objects reside) to represent the spatial attributes, while their second strategy utilizes a Bloom filter [18] representation of the partition and where objects are located. Both are applied in a general distributed spatial query processing strategy. Osborn and Zaamout [16] propose a general strategy for distributed spatial join processing that considers multiple sites and minimizes data transmission costs. Their strategy begins by identifying the smaller participating relations, before shipping the spatial attribute from each to a site with a larger relation. A semi-join-based join is performed on the larger relations, with the qualifying identifiers shipped back to the sites of the smaller relations in order to select qualifying tuples. All qualifying tuples are then sent to a central query site for final spatial join processing.

Recently, two strategies have been proposed for spatial query processing on spatial data streams. The progressive spatial join strategy proposed by Kwon and Li [3] works by joining the same pair of objects multiple times using different representations of them, beginning with a minimum bounding box, and increasing the complexity of the representation for

subsequent joins This reduces the cost of the spatial join as simpler representations are computationally less expensive to join, and can also prune object pairs that cannot be joined [3]. Osborn [17] proposed several strategies that use a conventional spatial join, or a bit-array join (i.e., similar to a Bloom join [18]). All strategies utilize a common region (i.e., common spatial extent) between the two spatial data streams, in order to prune objects that would not be part of the spatial join.

Therefore, significant work has taken place for applying spatial joins in various strategies to centralized and distributed query processing, with very little work having taken place in applying spatial joins in stream query processing. In addition, existing strategies have one or more of the following limitations:

- the assumption that all participating objects are available for the joins (i.e., an object cannot be transmitted at a later time and still be considered "current" for the spatial join.
- the repeated spatial join of the same pair of objects.
- the spatial extents of the individual spatial data streams must be known in advance. It is not always possible to know this latter requirement – at least not accurately.

Therefore, two new strategies are proposed in this paper that focus on overcoming these limitations. Both strategies do not require that the objects be available at the time of the join, and that the spatial extent from which the spatial data stream objects are drawn be known in advance. In addition, they do not perform repeated spatial joining of the same two objects. Both strategies are proposed in the next section.

3. Background

This section presents the required background for understanding the proposed work. After introducing the symbolic notation used in this paper, the terms spatial data stream system, sliding window, and spatial join are defined. Finally, the *Spatial2* strategy [17] is summarized, which is the strategy used for comparison in Section 5.

3.1. Symbolic Notation

The following symbols are used throughout this paper:

- S_1 and S_2 are two spatial data streams which transmit objects to the spatial data stream server.
- o_1 and o_2 are the current objects arriving from S_1 and S_2 respectively.
- E_1 and E_2 are the spatial extents that contains objects S_1 and S_2 respectively.
- *CR* is a **common region** of space shared by both S_1 and S_2. It is composed from the overlap of E_1 and E_2.
- *RS* is the result stream (i.e., output stream).
- *SW* is a sliding window.
- *numobj(SW)* depicts the size of *SW* (or, the number of objects currently in *SW)*.
- o_{swx}, x = 1 to *numobj(SW)* are the objects that have been placed in *SW* from both S_1 and S_2.

3.2. Data Stream System

Fig. 1 depicts an example data stream system. There are two input streams, which continuously create and transmit data to a server, which serves as the stream data manager and processor. For the work to be proposed here, it is assumed that the server contains a query processor that processes data immediately on its arrival. Any generated results from query processing are transmitted on the result data stream to some other remote destination [2]. For spatial data streams, all data that is generated, transmitted, and produced on the server is spatial data, which is some combination of point and/or object data that has a location in space.

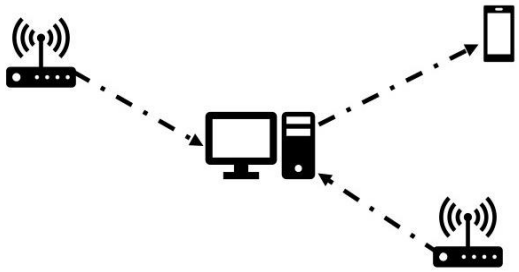

Fig. 1. Data stream system

Since a limited amount of data that arrives from a data stream can be stored in stream memory at a time, decisions must be made concerning which data to store and which to delete. Several approaches have been proposed for this [1]. One approach for managing incoming stream data is called a **sliding window**. Conceptually, it is a view of a subsequence of the data at a given time, and after the subsequence is processed, this view is moved along the sequence to process other subsequences. In a data stream system, a sliding window can be seen as a limited size memory that stores incoming stream data for some finite amount of time. When the sliding window is full, some data must be removed in order to accommodate more incoming data from the stream. By definition, a sliding window uses a first-in-first-out (FIFO) strategy – in other words, the data that has remained in the sliding window the longest is chosen for removal. The work proposed here utilizes the FIFO strategy for object removal. However, this work also utilizes the least recently used (LRU) strategy. The LRU strategy determines which objects have not been accessed in the longest time (e.g., an object for a spatial join) and chooses these objects for removal. Both strategies are employed to determine if one approach is superior with respect to keeping the most needed data in the sliding window at all times.

Traditionally, a data stream is defined as **unbounded** if the number of values arriving on the stream was unknown [1]. For a spatial data stream, the term unbounded requires an additional component. In addition to the number of objects being unknown, the region of space (i.e., spatial extent) that the objects are drawn from can also be unknown. Therefore, a spatial data stream is unbounded if the number of objects is unknown, and/or its spatial extent is unknown.

3.3. Spatial Join

The spatial join [4] matches pairs of objects from spatial object sets, S_1 and S_2 based on the outcomes of some spatial predicate such as overlap, containment, direction, etc. The following assumptions are made for this work. First, the strategies proposed here assume the use of the overlap predicate in the spatial join. Second, this work also assumes that the nested-loop join is utilized for joining portions of S_1 and S_2. Third, the strategies assume that a vector (i.e., geometric) representation of polygons is utilized – therefore, a polygon is a sequence of connected points. Finally, it is assumed that all objects are in rectangular form only, as testing the overlap of any arbitrarily shaped objects is costly. However, the strategies proposed in this paper will work for any non-directional predicates, and spatial objects of any shape.

3.2. The Spatial2 strategy

The *Spatial2* [17] strategy will serve as the comparison strategy in the evaluation versus the new strategies (Section 5). *Spatial2* achieved high accuracy over the strategy in [3] when compared using spatial data streams with smaller numbers of objects [17]. Given two spatial data streams S_1 and S_2, the strategy processes the object inputs o_1 and o_2 as they arrive using the spatial join. Sliding window *SW* stores the most recent set of objects o_{swx} that have arrived from both streams and were chosen to be maintained. To decide which objects to keep, *Spatial2* utilizes the common spatial extent (i.e., common region *CR)* to determine the region of space that contains objects from both S_1 and S_2.

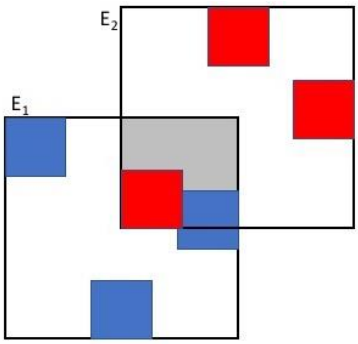

Fig. 2. Overlap of two overall regions

Fig. 2 depicts an example of a common region. Given spatial extents E_1 and E_2 which correspond to streams S_1 and S_2 respectively, the grey region is the spatial extent (i.e., common region *CR*) that potentially contains objects generated for both streams S_1 and S_2. *Spatial2* evaluates arriving objects o_1 and o_2 for overlap with *CR*, to determine: 1) if there is the potential for overlap with other objects, and 2) whether the object should be stored in *SW*. The strategy is carried out in the following manner [17].

1. Spatial extents E_1 and E_2 are transmitted to spatial stream query processor for calculating common region *CR*.
2. Spatial data streams S_1 and S_2 transmit objects o_1 and o_2 respectively to the data stream processor. If required, at most two objects must be removed from *SW* if no room exists for the new objects. *Spatial2* utilizes the FIFO strategy for object removal.
3. Object o_1 is tested for overlap with *CR*. If overlap exists, o_1 is first added to *SW*, and then evaluated for overlap with all other objects o_{swx} in *SW* that were transmitted on S_2. Any overlapping pairs that are found, are transmitted on the result stream *RS*.
4. Object o_2 is tested for overlap with *CR*. If overlap exists, o_2 is first added to *SW*, and then evaluated for overlap with all other objects o_{swx} in *SW* that were transmitted on S_1. Any overlapping pairs that are found, are transmitted on the result stream *RS*.
5. This is repeated from Step 2 while S_1 and S_2 continue transmitting objects.

4. Strategies for Unbounded Data Streams

In this section, two nested-loop strategies for joining two spatial data streams are proposed, called Sliding Window and Incremental. Both strategies do not require the existence of pre-determined spatial extents, as these may not be available, or require significantly overestimation in order to be applied to existing strategies. Both Sliding Window and Incremental are proposed below.

4.1. Sliding window strategy

The Sliding Window strategy creates a common region *CR* by estimating spatial extents E_1 and E_2 for the spatial data streams S_1 and S_2 respectively. These are estimated using the first *numobj(SW)/2* objects from each stream. The *CR* is created by performing the overlap of E_1 and E_2. Then, subsequent objects o_1 and o_2 that are transmitted from S_1 and S_2 are tested for overlap with *CR*. Only those objects that overlap with *CR* are saved in *SW* for future joins. The strategy proceeds as follows:

1. The first *numobj(SW)/2* objects from each of S_1 and S_2 are obtained. At the same time, any existing spatial joins are processed:
   a. Objects o_1 and o_2 are received from streams S_1 and S_2 respectively, and are placed on *SW*. If *SW* is full, either FIFO or LRU selection take places to remove existing objects.
   b. Object o_1 is tested for overlap with all other objects o_{swx} in *SW* received from S_2. Any pair of objects that overlap are transmitted on *RS*.
   c. Object o_2 is tested for overlap with all other objects o_{swx} in *SW* received from S_1. Any pair of objects that overlap are transmitted on *RS*.
2. Next, E_1, E_2 and *CR* are created:
   a. E_1 is formed by enclosing objects o_{swx} in *SW* that were transmitted from S_1.
   b. E_2 is formed by enclosing objects o_{swx} in *SW* and were transmitted from S_2.
   c. Finally, *CR* is created by performing the overlap of E_1 and E_2.
3. At this point, the *Spatial2* strategy is utilized to process joins with subsequent incoming objects from S_1 and S_2, and to determine which objects are to be added to and removed from *SW*.

4.2. Incremental strategy

The Incremental Strategy creates *CR* gradually as objects o_1 and o_2 continue to arrive from S_1 and S_2. An initial *CR* is created by encompassing the first o_1 and o_2 with a spatial extent. Then, for all subsequent o_1 and o_2, if o_1 and/or o_2 overlap *CR*, then *CR* is increased the accommodate the new object(s), and the objects are added to *SW* (with space being made in *SW* for them if required). Spatial joins with other objects in *SW* are also processed at this point. The strategy proceeds as follows:

1. *CR* is created initially by creating a spatial extent that encompasses o_1 and o_2.
2. Then, all subsequent objects o_1 and o_2 are processed as they arrive from S_1 and S_2:
   a. Object o_1 is tested for overlap with the *CR*. If successful, *CR* is increased to accommodate o_1, and o_1 is added to *SW*. In addition, o_1 is evaluated for overlap with all other objects o_{swx} in *SW* that were transmitted from S_2. Any pair that have a positive overlap result is transmitted on the result stream *RS*.
   b. Object o_2 is tested for overlap with the *CR*. If successful, *CR* is increased to accommodate o_2, and o_2 is added to *SW*. In addition, o_2 is evaluated for overlap with all other objects o_{swx} in *SW* that were

transmitted from S_1. Any pair that have a positive overlap result is transmitted on the result stream *RS*.

3. If required, if *SW* does not have enough room for o_1 and o_2, then one or two objects are chosen for removal.

5. Evaluation

In this section, the empirical evaluation of the Sliding Window and Incremental strategies is presented. Both strategies will be compared against the *Spatial2* strategy [17]. The methodology is presented first, followed by the presentation of the results of the evaluation.

5.1. Methodology

The simulated data streaming environment set up for all experiments contains:

- two simulated spatial data streams, and
- a central stream query processor with one sliding window.

The sliding window manages a portion of both spatial data streams. It also utilizes the first-in-first-out (FIFO) and least recently used (LRU) object removal when space was needed in the sliding window for new incoming objects.

Both strategies are implemented in C++ on the Linux Centos 7 operating system. For evaluation purposes, several spatial data streams were simulated by using a sequence of *10x10* rectangles. This approach was used so that the distribution of the data, the size of the spatial extent covered by a spatial data stream, and the overlap between spatial extents, could be controlled. Altogether, 20 simulated spatial data streams were used in pairs for spatial joins and grouped into two evaluation sets. The first used pairs containing 2000, 4000, 6000, 8000 and 10000 rectangles, respective. The second used pairs containing 20000, 40000, 60000, 80000 and 100000 rectangles, respectively. Each set of *n* rectangles were created from a spatial extent of dimensions:

$$(\text{sqrt}(n) * 10) \text{ x } (\text{sqrt}(n) * 10)$$

For example, a *310x310* spatial extent was used to create the 1000-rectangle sets. In addition, between pairs of rectangle sets, the spatial extents had a 25% overlap between them.

For all strategies, two sets of tests were carried out that varied: 1) the number of objects sent through each spatial data stream, and 2) the size of the sliding window. For each set of tests:

- *Varying the number of objects.* Overall, 20 tests were carried out, which can be categorized into five groups:
  - *2000x2000*, *4000x4000*, *6000x6000*, *8000x8000* and *10000x10000* spatial object stream pairs, with sliding window size 16000 bytes (i.e., 1000 rectangles),
  - *20000x20000*, *40000x40000*, *60000x60000*, *80000x80000* and *100000x100000* spatial object stream pairs, with sliding window size 16000 bytes (i.e., 1000 rectangles),
  - *2000x2000*, *4000x4000*, *6000x6000*, *8000x8000* and *10000x10000* spatial object stream pairs, with sliding window size 160000 bytes (i.e., 10000 rectangles),
  - *20000x20000*, *40000x40000*, *60000x60000*, *80000x80000* and *100000x100000* spatial object stream pairs, with sliding window size 160000 bytes (i.e., 10000 rectangles),
- *Varying the window size.* Overall, 16 tests were carried out, which can be categorized into five groups:
  - 8000, 16000, 24000 and 32000 bytes (i.e., 500, 1000, 1500 and 2000 rectangles, respectively), using the *4000x4000* spatial object stream pair.
  - 8000, 16000, 24000 and 32000 bytes (i.e., 500, 1000, 1500 and 2000 rectangles, respectively), using the *40000x40000* spatial object stream pair.
  - 80000, 160000, 240000 and 320000 bytes (i.e., 5000, 10000, 15000 and 20000 rectangles, respectively), using the *4000x4000* spatial object stream pair.
  - 80000, 160000, 240000 and 320000 bytes (i.e., 5000, 10000, 15000 and 20000 rectangles, respectively), using the *40000x40000* spatial object stream pair.

For all tests, two performance factors were recorded: the CPU time (in milliseconds) over the entire join at the stream query processor, and the number of joined tuples in the final result stream. Accuracy was calculated from the latter value using number of tuples in the overall join result (i.e., assuming a full spatial join with no sliding window) that was also determined in each strategy.

5.2. Results

This section presents the results of the evaluation. The results for the varying spatial data stream sizes are presented first, followed by the results for the varying sliding window sizes. In all charts (Figs. 3-18), the following is represented: *CR* is the outcome of the *Spatial2* strategy [17], *SW FIFO* is the outcome of the Sliding Window strategy where a FIFO strategy is utilized for object removal from *SW*, *Inc FIFO* represents the outcome of the Incremental strategy where a FIFO strategy is utilized for object removal from *SW*, *SW LRU* represents the outcome of the Sliding Window strategy where a LRU strategy is utilized for object removal from *SW*, and *Inc LRU* represents the outcome of the Incremental strategy where a LRU strategy is utilized for object removal from *SW*.

5.2.1. Varying Number of Objects

Figs. 3 to 6 depict the accuracy results for varying the size of the spatial data streams while using sliding windows of 16,000 bytes and 160,000 bytes respectively. As we can see, results show that overall, there is a significant decrease in accuracy as the number of objects being transmitted from the data streams increase. In addition, with respect to accuracy, the Sliding Window strategy performs significantly better than the Incremental Strategy. Finally, we also see there is little to no difference between the use of FIFO and LRU for selecting which objects to remove from the sliding window.

In Figs 3 and 4, the accuracy of the Sliding Window strategy is high and almost equal to that achieved by the *Spatial2* strategy, and for smaller data sizes is at least 80%. This is significantly better than the Incremental strategy, which at best only achieves a 60% accuracy. Both strategies, however, have a significant decrease in accuracy as the number of objects increase, with less than 20% in the larger stream sizes. With an increase in the sliding window size, which is shown in Figs. 5 and 6, improvements - significant ones early on, and more modest one with larger stream sizes - are achieved by the Sliding Window strategy. However, again it must be noted that the Sliding Window strategy performs almost as well as the *Spatial2* strategy. Therefore, having a pre-determined spatial extent from which objects are streamed from is not necessary, and comparable results are achieved.

Figs. 7 to 10 depict the running time results for varying the size of the spatial data streams while using sliding windows of 16,000 bytes and 160,000 bytes respectively. As we can see,

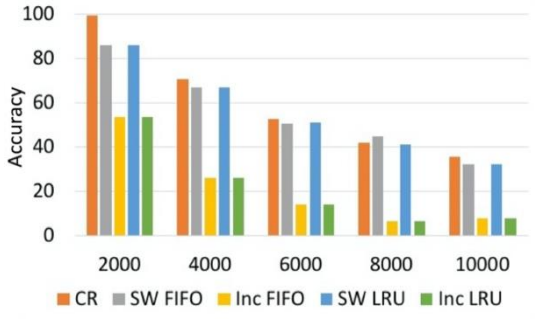

Fig. 3. Up to 10000x10000 objects – window size 16K (acc)

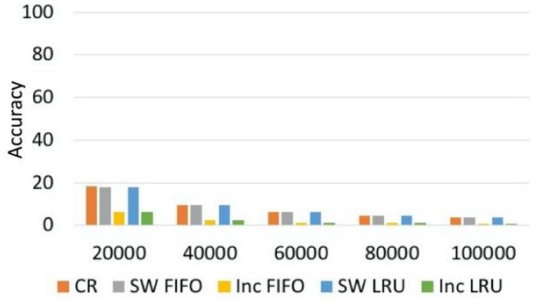

Fig. 4. Up to 100000x100000 objects – window size 16K (acc)

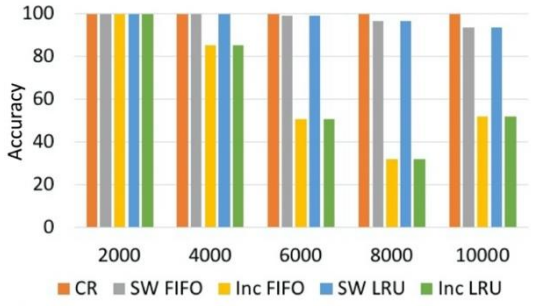

Fig. 5. Up to 10000x10000 objects – window size 160K (acc)

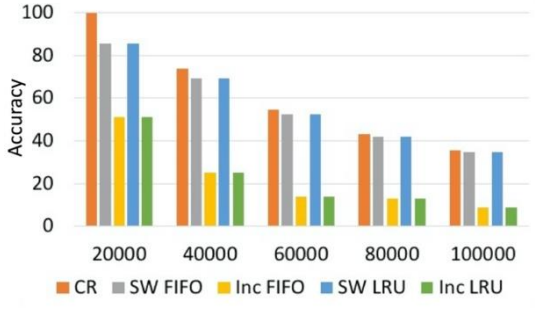

Fig. 6. Up to 100000x100000 objects – window size 160K (acc)

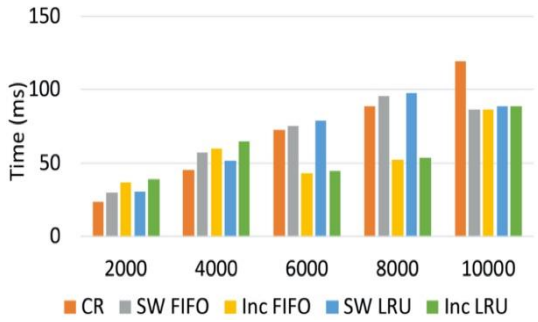

Fig. 7. Up to 10000x10000 objects – window size 16K (time)

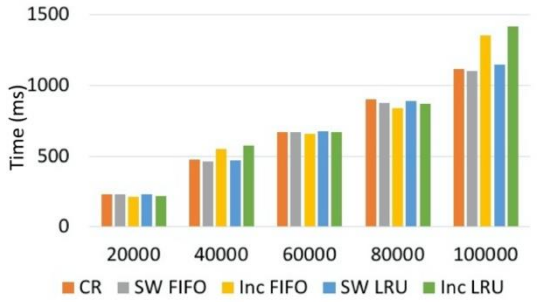

Fig. 8. Up to 100000x100000 objects – window size 16K (time)

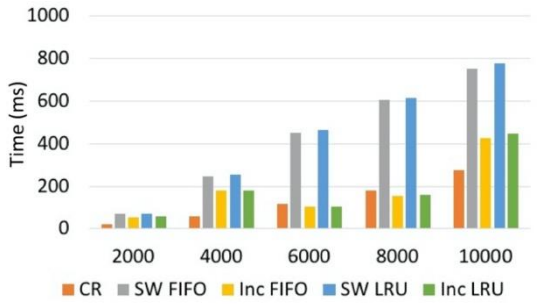

Fig. 9. Up to 10000x10000 objects – window size 160K (time)

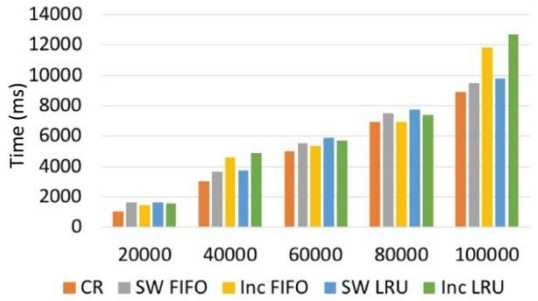

Fig. 10. Up to 100000x100000 objects – window size 160K (time)

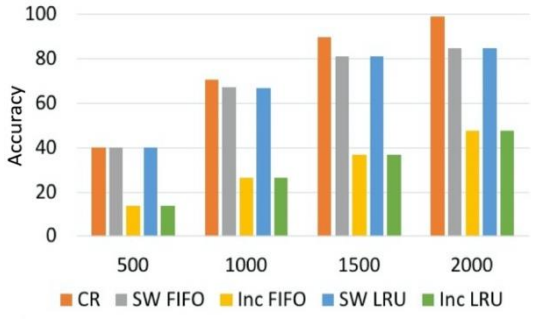

Fig. 11. Up to 32K, 4000x4000 objects (acc)

Fig. 12. Up to 32K, 40000x40000 objects (acc)

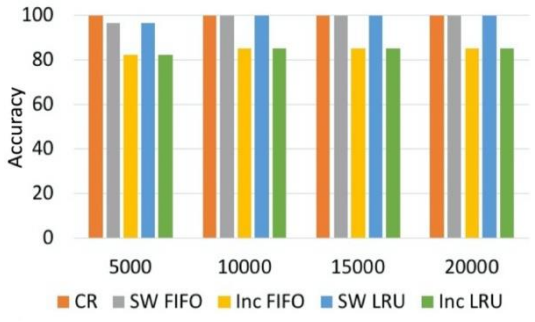

Fig. 13. Up to 320K, 4000x4000 objects (acc)

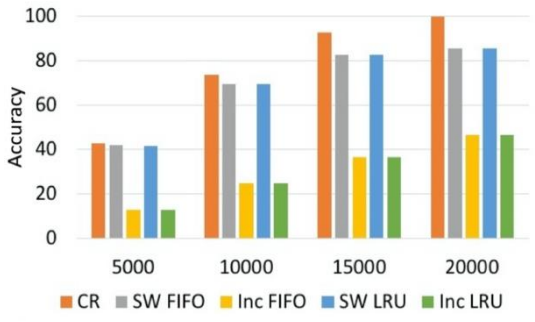

Fig. 14. Up to 320K, 40000x40000 objects (acc)

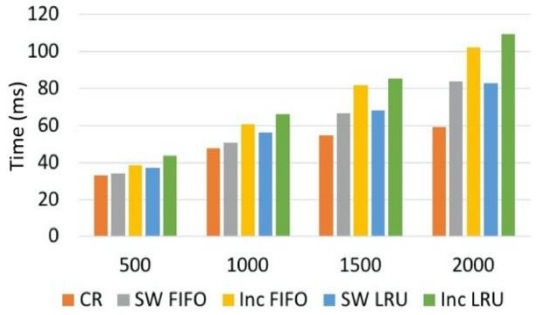

Fig. 15. Up to 32K, 4000x4000 objects (time)

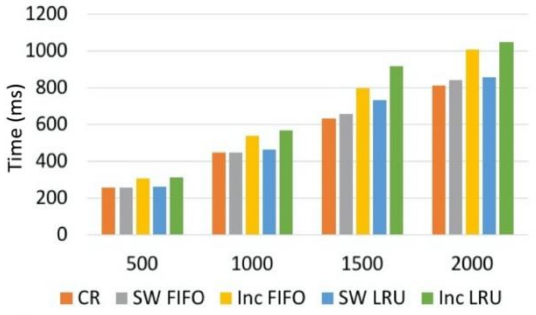

Fig. 16. Up to 32K, 40000x40000 objects (time)

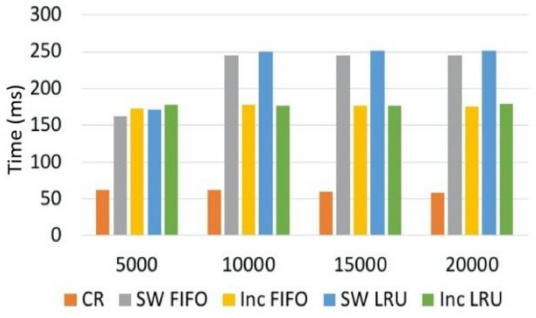

Fig. 17. Up to 320K, 4000x4000 objects (time)

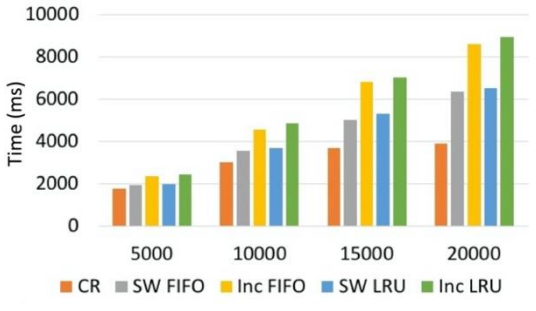

Fig. 18. Up to 320K, 40000x40000 objects (time)

results show a significant increase in running time as the size of the data streams increase. In addition, the Incremental strategies – both with FIFO and LRU object removal - have an initial lower running time over the other approaches, but ultimately produce a higher running time as the number of objects increase.

5.2.2. Varying Sliding Window Size

Figs. 11 to 14 depicts the accuracy results for varying the size of the sliding window with spatial join sizes of 4000x4000 and 40000x40000 respectively. From the results, we see that for larger data sizes, the accuracy does improve as the size of the sliding window increases, but overall, the size of the dataset requires a larger sliding window size in order to achieve better performance. We also see that there is little to no difference between the use of FIFO and LRU for selecting which objects to remove from the sliding window. We see similar trends to those found for varying the number of objects transmitted from the data streams. Specifically, we see that the Sliding Window Approach achieves accuracies that are similar to those obtained by the *Spatial2* strategy, and that the Incremental strategy performs poorly. The only exception is for the larger sliding windows and the *4000x4000* spatial stream join, where the Incremental strategy does achieve 80% accuracy.
Figs. 15 to 18 depict the accuracy results for varying the sliding window size, with spatial join sizes of 4000x4000 and 40000x40000 respectively. Results show that the running time increases as the size of the sliding window increases.

We also observe the same trends as above - that the Sliding Window strategy has higher running times than the Incremental strategy for the smaller sliding window sizes, but the opposite occurs as both the sliding window size and the size of the spatial data stream increase.

5.3. Assumptions and Threats to Validity

This section presents assumptions and potential threats to the validity of the experimental results.
The objects used for all experiments are squares in which all vertices are double-precision floating point numbers. No challenges exist here, as long as non-numeric data is not used to represent the vertices. In addition, although the identifiers associated with each object are integers, they are represented as strings. Therefore, any other representation of the identifier should not cause problems with program execution (although this was not specifically verified). However, any non-rectangular representation of objects will affect these results, as the implementation of the strategies only test for overlap of rectangles.
In a sequence of objects, all objects are unique with respect to location (i.e., no two objects have the same vertices). Although having two or more completely identical objects in a stream is very unlikely, the possibility is higher here as all objects have an identical representation. In addition, although the execution of the problem is not affected by having duplicate objects, the results obtained here may vary if this situation existed.
Both streams send objects to the server at the exact rate and time. Any changes to this will affect the results obtained here.
Finally, the strategies consider heuristically both the validity of the input with respect to its potential for participation in a join, and whether an object will be required later on. The experiments results may differ if the objects arrive in a different order. In addition, the strategies assume that all outputs (i.e., spatially joined pairs that are transmitted on an output stream) are valid forever. Consideration of invalidity of outputs will affect the results obtained here.

6. Conclusion

In this paper, two strategies for spatial join processing in spatial data streams – Sliding Window and Incremental – are proposed. The goals of both strategies are: 1) to eliminate the need for up-front spatial extents from the participating spatial data streams, by estimating the common spatial extent shared by the streams, and 2) to prune objects that have little or no chance to participate in the spatial join. The first strategy, Sliding Window, estimates the common region by using the first *numobj(SW)* objects that arrive from the spatial data streams. The second strategy, Incremental, forms an initial common region from the first objects - one from each stream - that arrive, and grow the common region incrementally with subsequent arriving objects that overlap the current common region.
An experimental evaluation and comparison versus the *Spatial2* approach [17] that uses pre-determined spatial extents show that the Sliding Window approach performs significantly better than the Incremental approach and, in terms of accuracy, almost equal to the *Spatial2* approach.

6.1. Future Work

Future research directions include the following. First, spatial data streams with differing arrival times must be considered, as currently both are transmitting objects at the exact same rate and time. In addition, other strategies for estimating the spatial extent of an unbounded spatial data stream must be considered, as certain accuracies – especially for larger spatial data streams and smaller sliding windows – are still too low. Finally, the strategies need to be evaluated with larger spatial data streams, and also against the original progressive join algorithm proposed by Kwon and Li [3].

References

[1] Babu S, Widom J. Continuous Queries over Data Streams. SIGMOD Record 2001;30:109-120. https://doi.org/10.1145/603867.603884

[2] Han J, Kamber M, Pei J. Data Mining: Concepts and Techniques. Massachusettes: Morgan Kaufmann, 2011.

[3] Kwon O, Li KJ. Progressive spatial join for polygon data stream. Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. Chicago, IL, 2011. https://doi.org/10.1145/2093973.2094030

[4] Shekhar S, Chawla S. Spatial databases: a tour. New Jersey: Prentice Hall, 2003.

[5] Patel J, Dewitt D. Partition based spatial-merge join. Proceedings of the 1996 ACM Sigmod International Conference on Management of Data. Montreal, QC, 1996. https://doi.org/10.1145/233269.233338

[6] Huang YW, Jing N, Rundensteiner E. Integerated query processing strategies for spatial path queries. Proceedings of the 13th IEEE International Conference on Data Engineering. Birmingham, UK, 1997.

[7] Zhou X, Abel D, Truffet D. Data partitioning for parallel spatial join processing. Geoinformatica 1998;2:175-204. https://doi.org/10.1023/A:1009755931056

[8] Arge L, Procopiuc O, Ramaswamy S, Suel T, Vitter J. Scalable sweeping-based spatial join. Proceedings of the 24th International Conference of Very Large Databases. New York City, NY, 1998.

[9] Jacox E, Samet H. Spatial join techniques. ACM Transactions on Database Systems 2007;32:43 pages. https://doi.org/10.1145/1206049.1206056

[10] Zhong Y, Han J, Zhang T, Li Z, Fang J, Chen G. Towards parallel spatial query processing for big spatial data. Proceedings of the 26th IEEE International Parallel and Distributed Processing Symposium Workshops. Shanghai, China, 2012. https://doi.org/10.1109/IPDPSW.2012.245

[11] Abel D, Ooi B, Tan KL, Power R, Yu J. Spatial join strategies in distributed spatial dbms. Proceedings of the 45th International Symposium on Advances in Spatial Databases. Portland, ME, 1995. https://doi.org/10.1007/3-540-60159-7_21

[12] Tan KL, Ooi B, Abel D. Exploiting spatial indexes for semijoin-based join processing in distributed spatial databases. IEEE Transactions on Knowledge and Data Enginnering 2000;12:920-937. https://doi.org/10.1109/69.895802

[13] Kalnis P, Mamoulis N, Bakiras S, Li X. Ad-hoc distributed spatial join on mobile devices. Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium. Rhodes Island, Greece, 2006. https://doi.org/10.1109/IPDPS.2006.1639266

[14] Karam O, Petry F. Optimizing distributed spatial joins using R-trees. Proceedings of the 43rd ACM Southeast Conference. Kennesaw, GA, 2005. https://doi.org/10.1145/1167350.1167417

[15] Farruque N, Osborn W. Efficient distributed spatial semijoins and their application in multiple-site queries. Proceedings of the 28th IEEE International Conference on Advanced Information Networking and Applications. Victoria, BC, 2014. https://doi.org/10.1109/AINA.2014.132

[16] Osborn W, Zaamout S. Using spatial semijoins over multiple sites in distributed spatial query processing. Canadian Journal of Electrical and Computer Engineering 2016;39:71-81. https://doi.org/10.1109/CJECE.2015.2463753

[17] Osborn W. Exploring bit arrays for join processing in spatial data streams. Proceedings of the 22nd International Conference on Network-based Information Systems. Oita, Japan, 2019. https://doi.org/10.1007/978-3-030-29029-0_7

[18] Bloom BH. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM 1970;13:422-426. https://doi.org/10.1145/362686.362692