Exploring Bit Arrays for Join Processing in Spatial Data Streams

Wendy Osborn(✉)

University of Lethbridge,
4401 University Drive West, Lethbridge, AB T1K 3M4, Canada
wendy.osborn@uleth.ca

Abstract. In this paper, the use of bit arrays for processing spatial joins in spatial data streams is explored. Although spatial joins between objects have been explored in other contexts, such as centralized and distributed systems, they have not been explored in great detail in spatial data streams. This work explores the use of a Bloom-filter (i.e., bit array) inspired representation of a spatial object. Strategies for both mapping objects to bit arrays, and processing spatial joins using the bit arrays in a data stream environment are presented. The strategies are evaluated and compared with spatial (non-bit) join approaches. Performance improvements are identified, and areas of improvement are also identified.

1 Introduction

Nowadays, applications exist that generate data in a continuous stream, where the amount of data cannot be stored in its entirety, and also may lose its validity after a certain amount of time [4,7]. Data streams require different strategies for handling them, as conventional strategies were designed for stored data [4]. This also extends to spatial data streams, where the streaming data consists of points or objects of non-zero area [15]. One type of strategy that needs to be re-visited is the spatial join [18] on two or more spatial data sets. Conventional spatial join processing requires that all objects are present for the join. Now, spatial objects are arriving as spatial data streams and therefore must be handled as they arrive. In addition, decisions must be made as to which objects to keep [7].

The spatial join has been applied previously in the research literature. Existing strategies can be classified into the following categories: spatial joins in a centralized system [2,3,9,10,17,20,21], a distributed system [1,6,8,11–14,16,19] and a streaming data system [15]. Although a significant amount of study has gone into applying spatial joins to queries centralized and distributed systems the main limitation in most scenarios is that both sets of spatial objects must available in their entirety when applying a spatial join between them. The only existing work that studied spatial joins in a spatial data stream environment is that of Kwon and Li [15]. They proposed a progressive join strategy where objects are joined multiple times using various levels of granularity in order to

L. Barolli et al. (Eds.): NBiS-2019, AISC 1036, pp. 73–85, 2020.
https://doi.org/10.1007/978-3-030-29029-0_7

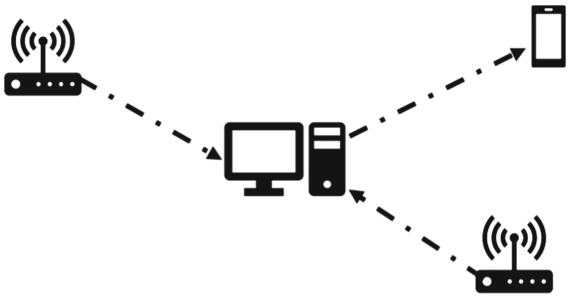

Fig. 1. Sensor network

reduce the cost of the spatial join. A limitation of this work is the multiple times the same two objects are processed.

Therefore, this paper proposes several strategies for processing a spatial join in a spatial data stream environment. In addition to proposing a strategy that uses a conventional spatial join, two additional strategies are proposed that utilize a bit array representation of a spatial object, in order to improve the time it takes to perform a spatial join, at the expense of some spurious results being generated. Two of the three strategies also apply a spatial join on a restricted portion of the space containing objects. An empirical evaluation and comparison versus simplified versions of the strategy in [15] shows that improvements in running time and accuracy can be achieved with the new strategies. In addition, some shortcomings of the new strategies lead to future research directions.

Section 2 provides some background that is required for the work to be proposed. In Sect. 3, the three strategies are proposed, and two simplified versions of the work of [15] are presented. Section 4 presents the methodology and results of the performance evaluation and comparison of the proposed strategies. Finally, Sect. 5 concludes the paper and provides some future research directions.

2 Background

In this section, the required background notably, spatial data stream system, spatial join and Bloom filter are summarized. Figure 1 depicts an example data stream. Here, multiple sensors (in the example, two) are generating data continuously, which is sent to a stream data processor. From here, any results are sent on a result stream to the required destination [7]. In a spatial data stream, the data being streamed is spatial data, which can be either point data, object data, or a combination.

A data stream processor can only store a certain amount of information from the streams. Several strategies have been proposed for choosing which data to keep [7]. One strategy is a sliding window, which is a memory of limited size that takes streaming data as it arrives and stores it for a limited time. When the sliding window is full, decisions are made as to which objects to remove. For the strategies in this paper, a first-in-first-out (FIFO) strategy is assumed.

A spatial join [18] takes two sets of spatial objects S_1 and S_2 and relates an object from each of the two sets using a spatial predicate, such as overlap, containment, and adjacency. The strategies proposed below assume the use of the overlap predicate, but with slight modification any other predicate can be utilized. Also assumed is the use of a nested-loop strategy for processing a join of a portion of each set spatial objects. Two different representations of a set of spatial objects are used in the proposed strategies below: a geometric (i.e., vector) representation, where each polygon is represented using points connected with lines; and a bit array representation, where a sequence of **1** and **0** represents the entire set of spatial objects.

The bit array utilized in the proposed strategies below is inspired by the Bloom filter [5]. A Bloom filter is a bit array that is generated by hashing some of the bit locations to **1**, while the other bit locations are left as **0**. They can be used to provide a compact representation of a set of values, such as an attribute in a database relation. If the Bloom filters of two or more attributes are created using the same hashing functions, then the filters can be bit-wise-anded to find potential common attribute values. Because hashing may produce the same bit addresses for two or more attribute values, some false positives may occur in other words, the resulting Bloom filter may indicate that a value exists in both attributes, when in fact this is not the case. Therefore, additional testing is necessary to eliminate false positives. The bit array approach that is utilized by the proposed strategies is presented in Sect. 3.

3 Stream Spatial Join Strategies

In this section, several strategies are proposed for join processing in spatial data streams two which utilize a bit array representation, and one that utilizes the original polygon representation. In addition, two polygon-based strategies will be summarized that will be used for comparison in Sect. 4. First, the approach for creating the bit array will be proposed. Then, the strategies Bit1, Bit2, and Spatial2 will be proposed, followed by a summarization of Spatial1 and Join.

For all strategies, where appropriate, the following notation will be utilized: S_1 and S_2 are the spatial data streams that provide objects or bit arrays to the data stream processor; E_1 and E_2 are the regions that contain objects from S_1 and S_2; OR is denoted as an "overall region", which contains some combination of E_1 and E_2; m and n are the grid dimensions used to partition OR; RS is the result stream; SW is the sliding window; $numobj(SW)$ is the current number of objects or bit arrays maintained in SW; b_1 and b_2 are bit arrays representing objects o_1 and o_2 respectively; and o_{swx}, $x = 1$ to $numobj(SW) - 1$ are the objects or bit arrays in SW.

3.1 Bit Array Mapping

As mentioned above, the bit array is inspired by the Bloom filter [5]. However, the following differences are applied to their creation here:

- A separate bit array is created for each spatial object, instead of one bit array for the entire object set. Because each spatial object is being transmitted from a sensor one at a time, and may not remain in the sliding window for the duration of the join operation, it makes sense to create individual bit arrays over one bit array for the entire object set.
- The location for the **1** bits are chosen based the location of the object on a grid, instead of using hashing.

Therefore, the following is the proposed strategy for taking an object in polygon form and mapping it to a bit array representation. First, the region OR is formed by combining E_1 and E_2 and determining the minimum extent that encompasses both. OR is then partitioned into a grid of $n \times m$ cells. Following this, a bit array is created and initialized to all zeros. The size of the bit array is $m * n$, which means that each bit in the array corresponds to one location in the grid. For an object o_i, if any portion overlaps a cell, a *1* bit will be assigned to the corresponding bit in the bit array; otherwise the bit is left as *0*.

To determine if potential overlap exists between two objects using their bit arrays, this can be done by performing a simple bit-wise-and of the bit arrays. If the result of the bit-wise-and is "positive" (i.e., b_1 & $b_2 != 0$), then potential overlap exists between the objects.

Figure 2 depicts the mapping of the two rectangles (pictured as red and blue) into their respective bit arrays, and the result of a bit-wise-and operation. It should be noted that the mapping takes place in this example in column-major format however, this is not strictly required and using row-major format will not affect the result. It can be seen that overlap exists between the red and blue rectangles since the bit arrays have *1* bits in the same locations, which would produce a non-zero bit-wise-and result.

However, it is possible that the result of a mapping and bit-wise-and computation will produce a "positive" result that is a false positive (i.e., a true outcome that is actually false). Figure 3 depicts such a result. Both the red and blue rectangles map to bit arrays that produce a "positive" outcome when a bit-wise-and operation is applied to them. But we see that the rectangles do not overlap. This is referred to in this paper as a *spurious tuple* one that is generated even though, officially, it is not part of the result. Therefore, a trade-off exists, between (hopefully) achieving a faster result versus having extra tuples formed.

Now, the spatial data stream join processing strategies will be presented, beginning with the bit array strategies, then followed by the polygon strategies.

3.2 Bit1 Strategy

The **Bit1** strategy utilizes a bit array representation of every object that arrives from the spatial data streams. The sensors that generate objects must coordinate in order to determine the overall region OR so that all objects are properly mapped to bit arrays and can be compared by the spatial data stream processor. Then, the spatial stream module processes the spatial join in the following manner:

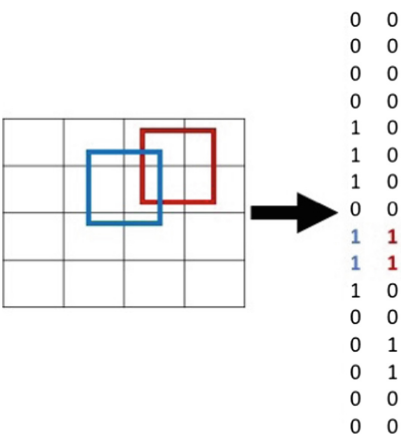

Fig. 2. Bit mapping - accurate result

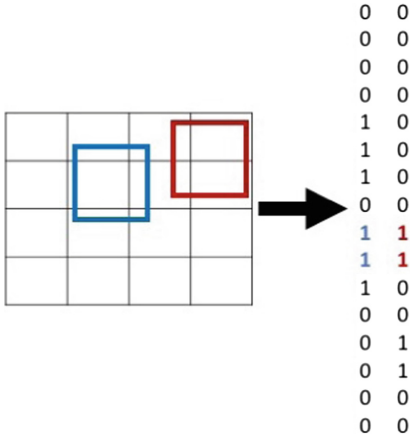

Fig. 3. Bit mapping - false positive

1. Bit arrays b_1 and b_2, representing objects o_1 and o_2 are received one from stream S_1 and the other from stream S_2.
2. First, if required, room must be made for the new bit arrays if the sliding window is full. Although any strategy can be used for selecting which bit arrays to remove, this work assumed a First-in-First-out strategy, which reflects that "older" bit arrays should be removed from the window.
3. The bit array b_1 is added to the sliding window. One at a time, b_1 is bit-wise-anded with all other bit arrays o_{swx} that are from S_2 and currently in SW. Any bit-wise-and operations that produce a "positive" result (i.e., b_1 & $o_{swx}! = 0$) are placed onto RS.
4. Next, the bit array b_2 is added to the sliding window. One at a time, b_2 is bit-wise-anded with all other bit arrays o_{swx} that are from S_1 and currently in SW. Any bit-wise-and operations that produce a "positive" result (i.e. b_2 & $o_{swx}! = 0$) are placed onto RS.
5. This process repeats from Step 1 until no more bit arrays are sent.

3.3 Bit2 Strategy

The *Bit2* Strategy also utilizes a bit array representation of objects. However, it improves upon the *Bit1* strategies in the following way. Figure 4 contains two overall regions E_1 and E_2, both with potential stream objects. Note the overlapped area (denoted from here as OR), which is highlighted in gray. One major improvement can be made to the *Bit1* strategy if we observe the following: Only objects that overlap with OR can potentially be part of the final join result. Therefore, only those objects that overlap OR need to be mapped into bit arrays! One other difference that must be noted is that the stream data processor performs the mapping of objects into bit arrays however, the mapping will only be with respect to OR, and it is expected that fewer objects needs to be mapped.

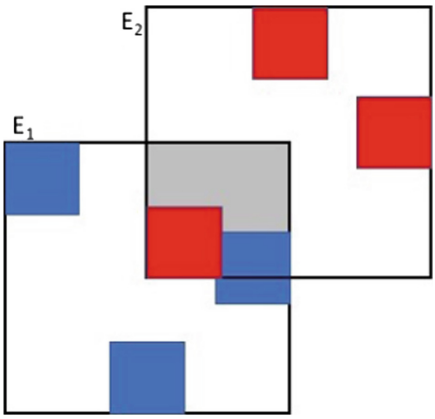

Fig. 4. Overlap of two overall regions

Given these modification, the spatial stream module of the **Bit2** Strategy processes the spatial join in the following manner:

1. First, the overall regions E_1 and E_2 are received from S_1 and S_2 by the stream query processor in order to determine OR.
2. Then, two objects o_1 and o_2 are received from S_1 and S_2.
3. If required, room must be made for the objects if the sliding window is full, using the First-in-First-out strategy.
4. If the object o_1 overlaps OR, it is mapped into b_1 and added to the sliding window. One at a time, b_1 is bit-wise-anded with all other bit arrays o_{swx} that are from S_2 and currently in SW. Any bit-wise-and operations that produce a "positive" result (i.e. b_1 & $o_{swx}! = 0$) are placed onto RS.
5. If the object o_2 overlaps OR, it is mapped into b_2 and added to the sliding window. One at a time, b_2 is bit-wise-anded with all other bit arrays o_{swx} that are from S_1 and currently in SW. Any bit-wise-and operations that produce a "positive" result (i.e. b_2 & $o_{swx}! = 0$) are placed onto RS.
6. This process repeats from Step 2 until no more bit arrays are sent.

3.4 Spatial2 Strategy

The *Spatial2* strategy is also proposed in this paper. Its main purpose here is to provide a comparison between utilizing the actual objects from a spatial data stream versus utilizing bit array representations. However, to the best of the author's knowledge, this approach has not be proposed before.

Spatial2 is an adaptation of *Bit2* that tests arriving objects o_1 and o_2 for overlap with OR, and instead of mapping o_1 and o_2 to bit arrays, the objects themselves are stored in SW. The strategy proceeds as follows:

1. First, the overall regions E_1 and E_2 are received from S_1 and S_2 by the stream query processor in order to determine OR.

2. Then, two objects o_1 and o_2 are received from S_1 and S_2.
3. If required, room must be made for the objects if SW is full, using the First-in-First-out strategy.
4. If the object o_1 overlaps OR, it is added to SW. One at a time, o_1 tested for overlap with all other objects o_{swx} that are from S_2 and currently in SW. Any overlap operations that produce a "positive" result (i.e., $o_1 \cap o_{swx}! = 0$) are placed onto RS.
5. If the object o_2 overlaps OR, it is added to SW. One at a time, o_2 tested for overlap with all other objects o_{swx} that are from S_1 and currently in SW. Any overlap operations that produce a "positive" result (i.e., $o_2 \cap o_{swx}! = 0$) are placed onto RS.
6. This process repeats from Step 2 until no more bit arrays are sent.

3.5 Spatial1 and Join Strategies

Finally, two strategies *Spatial1* and *Join* are presented. Their main purpose is for comparison. *Spatial1* is an adaptation of *Bit1* where, instead of receiving bit arrays from the data streams, the actual objects are received, and are managed and processed similarly to the *Bit1* strategy. *Join* is an adaptation of *Spatial1* where a sliding window of unlimited size is utilized for processing the spatial join. *Spatial1* and *Join* can be viewed as an adaptation of the progressive join approach proposed in [15], where only one level of object resolution is utilized.

4 Evaluation

In this section, the empirical evaluation of the Bit1, Bit2, and Spatial2 strategies is presented, along with a comparison against the Join and Spatial1 strategies. The framework and evaluation methodology is presented first. Then, the results of the evaluation and resulting discussion is presented.

4.1 Framework and Methodology

For all experiments, an environment was utilized that contained two simulated spatial data streams, with each containing a bit mapper. The central stream query processor utilizes a sliding window for maintaining a subset of objects or bit arrays that have arrived from the data streams. It also contains a bit mapper.

All of the strategies presented in Sect. 3 are implemented in C++ on a PC running Linux Centos 7. They were evaluated using several simulated spatial data streams that utilized synthetic sets of 10×10 rectangles. This approach was chosen so that certain characteristics such as the overall coverage area of a stream, as well as the overlap between the coverage areas of two streams, could be controlled. Altogether, eight sets of rectangles were utilized in pairs for spatial joins each pair contained 500, 1000, 1500 and 2000 rectangles, respectively. Each set of n rectangles is drawn from a region of space of dimension $(\sqrt{n} * 10) \times (\sqrt{n} * 10)$. For example, the rectangles in each of the 1000-rectangle set were

drawn from a 310×310 region of space. In addition, each pair of rectangles were created so that 25% of the overall region of space between them had overlap.

For all strategies covered in Sect. 3, three sets of tests were carried out that varied: (1) the number of objects sent through each data stream, (2) the size of the sliding window, and (3) the size of the grid used for mapping the bit arrays. For each set of tests:

- *Varying the number of objects.* Four tests were carried out, for the 500×500, 1000×1000, 1500×1500 and 2000×2000 spatial join pairs respectively. The grid size was set to 11×11 (i.e., the closest to 128 bits, without going past that value), and the window size was set at 16000 bytes (i.e., 1000 rectangles or 1000 128-bit arrays).
- *Varying the window size.* Four tests were carried out using 8000, 16000, 24000 and 32000 bytes (i.e., 500, 1000, 1500 or 2000 rectangles or 128-bit arrays) respectively. The 1000×1000 spatial join query was utilized, along with an 11×11 grid size.
- *Varying the grid size.* Finally, four tests were carried out, for the 8×8, 11×11, 13×13 and 16×16 grid sizes (i.e., equivalent to 64, 128, 192, and 256 bits) respectively. The 1000×1000 spatial join query was utilized, along with a 16000 byte window size.

For all tests, in addition to the final spatial join stream, two performance factors were recorded:

- The CPU time over the entire join at the stream query processor.
- The number of joined tuples in the final result. Given this value for each strategy and the final spatial joins streams, accuracy was determined by calculating the following:
  - The number of tuples in the Join result that also existed in the Spatial1, Spatial2, Bit1 and Bit2 results, respectively (i.e., true positives).
  - The number of tuples in the Bit1 and Bit2 results that did not exist in the Join result (i.e., spurious tuples, or false positives).

4.2 Results and Discussion

The running time results will be presented first, followed by the accuracy results.

Figures 5, 6 and 7 present the running time results from varying the data size, sliding window size, and grid size respectively. With respect to the size of the spatial data streams on the running time (Fig. 5), it is observed that although the running time of all proposed strategies increase with the amount of data, the increase is not as significant when compared to the Join result. The best performing strategies are *Spatial1* and *Bit2*, which both have the lowest increases. *Bit2* has a slightly higher running time, which is due to the extra time required for mapping polygons to bit arrays. However, this difference over *Spatial2* is not significant. Both *Spatial1* and *Bit1* have the least desirable performance out of the four strategies. However, *Bit1* has a lower running time due to the use of the bit-wise-and operation, which does run faster than an overlap comparison of two polygons (even as simple rectangles).

With respect to varying streaming window size (Fig. 6), it is observed that once again *Spatial2* and *Bit2* have the best and fairly consistent performance, independent of the sliding window size. This is likely due to the restriction to the shared overlap area for identifying candidate objects and bit array for spatial joins not as many objects are being processed, and also added to and removed from the sliding window. Again, *Bit2* had a higher running due to having to perform bit array mapping. Surprisingly, *Spatial1* and *Bit1* have poor performance as the sliding window size increases. An increase in the size of the sliding window also leads to an increase in the time to search it whenever new objects or bit arrays arrive at the spatial stream server. It was expected that a larger sliding window would result in decreases in time, but was not the case.

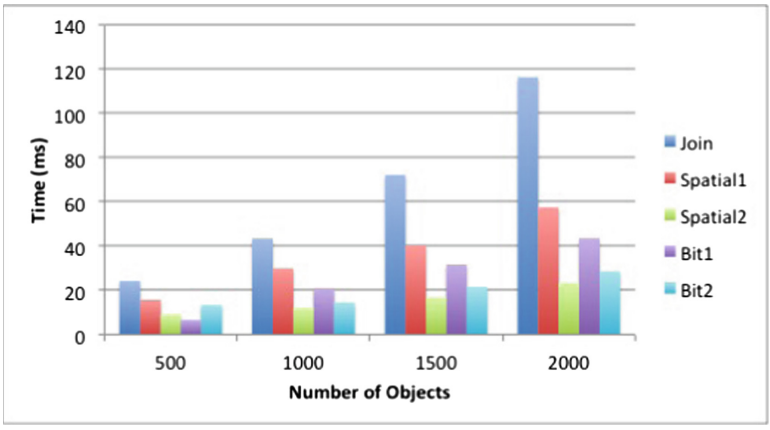

Fig. 5. Time for varying number of objects

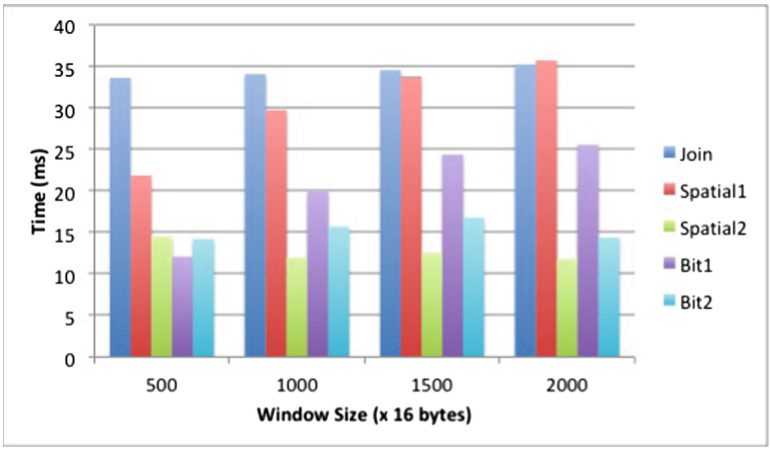

Fig. 6. Time for varying window sizes

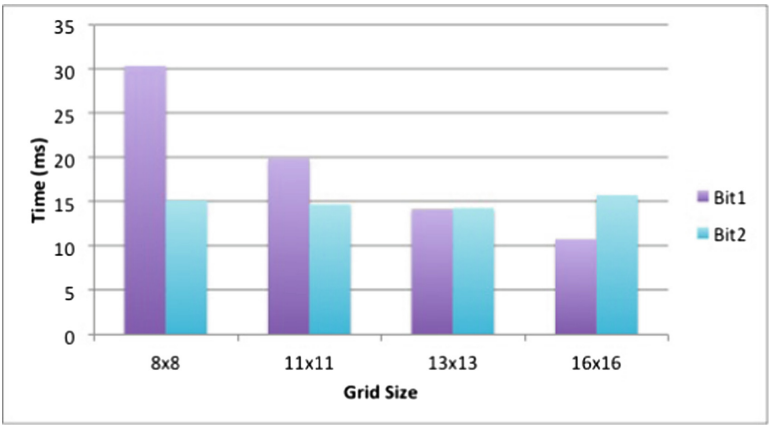

Fig. 7. Time for varying grid sizes

Finally, with respect to varying the grid size on the running time of *Bit1* and *Bit2* (Fig. 7), it is observed that for *Bit1* than an increase in the grid granularity actually leads to a decrease in running time! Again, the opposite was expected, but there is a reasonable explanation for this. As the grid size increases, the size of the bit array increases, which means that fewer bit arrays fit in the sliding window which leads to a lower running time for repeatedly searching the window. This trend is not observed for *Bit2*. However, the reason for this could be that the focus area is much smaller and therefore fewer bit arrays are being dealt with in general.

Tables 1, 2, and 3 present the accuracy results from varying the data size, sliding window size and grid size respectively. In all tables, the strategy names indicate the number of tuples in the final result, while %join indicates the accuracy of the strategy, and #spur indicates the number of spurious tuples that were generated by *Bit1* and *Bit2*. The accuracy will be discussed first, followed by the issue of spurious tuples.

With respect to the size of the spatial data streams on the accuracy of the results (Table 1) it is observed that, the best results were *Spatial2* and *Bit2*, both with 100% accuracy, with a slight decrease beginning at 2000 objects. This is a result of the combination of the window size (1000 objects or bit arrays, or 16000 bytes) and only consulting the restricted overlapped space. Very few if any

Table 1. Accuracy for varying data sizes

Data size	Join	Sp1	%join	Sp2	%join	Bit1	%join	#spur	Bit2	%join	#spur
500	421	421	100	421	100	2841	100	2420	1054	100	633
1000	952	712	74.79	952	100	8406	74.79	7694	2918	100	1966
1500	1455	845	58.08	1455	100	11268	58.08	10423	5613	100	4158
2000	1930	841	43.58	1922	99.59	14969	43.58	14128	7923	99.59	6002

Table 2. Accuracy for varying window sizes

Window size	Join	Sp1	%join	Sp2	%join	Bit1	%join	#spur	Bit2	%join	#spur
500	952	403	42.33	934	98.10	4832	42.33	4429	2862	98.10	1928
1000	952	712	74.79	952	100	8406	74.79	7694	2918	100	1966
1500	952	894	93.91	952	100	10428	93.91	9534	2918	100	1966
2000	952	952	100	952	100	11083	100	10131	2918	100	1966

Table 3. Accuracy for varying grid sizes

Grid size	Join	Bit1	%join	#spur	Bit2	%join	#spur
8×8	952	18581	100	17629	4201	100	3249
11×11	952	8406	74.79	7694	2918	100	1966
13×13	952	4790	54.94	4267	2630	100	1678
16×16	952	2838	42.33	2435	2250	98.21	1321

objects or bit arrays need to be removed in order to make room for new ones. Unfortunately, the same cannot be said for *Spatial1* and *Bit2*. As the number of objects in the spatial data stream increases, the accuracy decreases significantly down to just over 40%. This is a result of considering all objects in both streams participating in the spatial join.

With respect to varying the sliding window size on the accuracy of the results (Table 2), a similar outcome is seen again for *Spatial2* and *Bit2*, which is almost 100% accuracy, regardless of the size of the sliding window. For *Spatial1* and *Bit1*, it is observed that a larger sliding window significantly helps to improve accuracy, with 100% being achiever for the 2000×2000 case. With more objects being considered here, many are being moved out of the sliding window before future objects have arrived that require them for a join.

Finally, with respect to the grid size on the accuracy of the results (Table 3), we see a similar result again for *Bit2* as before. For *Bit1*, it is observed that the accuracy significantly decreases as the grid size increases. This is a result of the larger grid producing a larger bit array, which means fewer of them can be stored in the same sized sliding window. Therefore, more bit arrays would need to be removed from the sliding window in order to make room for more, and having some that were required later on not longer being available.

With respect to *Bit1* and *Bit2*, both are generating an unacceptable number of spurious tuples. Given that *Bit2* operates on a reduced overlapped space, the number being generated by it is significantly lower than the number being generated by *Bit1*. The issue is with the grid size. A very fine grained (i.e., large) grid will result in a lower number of spurious tuples, since the cells will be smaller. However, the drawback is that the bit array is larger and fewer of them will fit into the same sized sliding window as the actual objects themselves.

Overall, it is found that *Spatial2* and *Bit2* has the best performance, and the most consistent performance for both running time and accuracy, regardless of

the number of objects, sliding window size, and grid size. However, for *Spatial1* and *Bit1*, it is observed that for an increase in data size, there is an increase in running time but a significantly decrease in accuracy. Given an increase in the sliding window size, there is an increase in accuracy and running times, while an increase in grid size results in a decrease in both accuracy and running time. Therefore, reducing the space that is looked at between two spatial data streams results in significant performance improvements.

5 Conclusion

In this paper, three spatial join strategies for spatial data streams are proposed. Two utilize a bit array mapping strategy to speed up comparisons between spatial objects, while one utilizes a conventional spatial join between objects. Two of the three strategies operate on a reduced space where both spatial data streams overlap. An experimental evaluation and comparison versus modified versions of an existing progressive spatial join strategy shows that all three algorithms outperform the existing strategies with respect to running time, with the two that operate on reduced spatial outperforming all others with almost 100% accuracy regardless of the size of the data stream, sliding window, or grid.

This work has resulted in many future research directions. One very important one is determining how to relate the bit array back to the actual geometric object it is representing in a data stream environment, which is not being done at the moment. Other directions include: (1) reducing the number of spurious tuples being generated by all strategies, (2) With respect to the *Bit2* strategy, one improvement that could help further improve its running time is to have the bit arrays mapped on the sensor generating the spatial data, (3) considering spatial data streams where the coverage area is not known in advance, (4) considering spatial data streams where the arrival times of objects differ, (5) comparing the strategies with larger spatial data streams and against the original progressive join algorithm proposed by [15].

References

1. Abel, D., Ooi, B., Tan, K.L., Power, R., Yu, J.: Spatial join strategies in distributed spatial DBMS. In: Proceedings of the 4th International Symposium on Advances in Spatial Databases (1995)
2. Aji, A., Wang, F., Vo, H., Lee, R., Liu, Q., Zhang, X., Saltz, J.: Hadoop-GIS: a high-performance spatial data warehousing system over MapReduce. Proc. VLDB **6**, 1009–1020 (2013)
3. Arge, L., Procopiuc, O., Ramaswamy, S., Suel, T., Vitter, J.: Scalable sweeping-based spatial join. In: Proceedings of the 24th International Conference on Very Large Databases, pp. 570–581 (1998)
4. Babu, S., Widom, J.: Continuous queries over data streams. SIGMOD Rec. **30**(3), 109–120 (2011)
5. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun ACM **13**(7), 422–426 (1970)

6. Farruque, N., Osborn, W.: Efficient distributed spatial semijoins and their application in multiple-site queries. In: Proceedings of the 28th IEEE International Conference on Advanced Information Networking and Applications. IEEE Computer Society (IEEE) (2014)
7. Han, J., Kamber, M., Pei, J.: Data Mining: Concepts and Techniques. Morgan Kaufmann, Boston (2011)
8. Hua, Y., Xiao, B., Wang, J.: BR-tree: a scalable prototype for supporting multiple queries of multidimensional data. IEEE Trans. Comput. **58**(12), 1585–1598 (2009)
9. Huang, Y.W., Jing, N., Rundensteiner, E.: Integrated query processing strategies for spatial path queries. In: Proceedings of the 13th International Conference on Data Engineering, pp. 477 –486 (1997). https://doi.org/10.1109/ICDE.1997.582010
10. Jacox, E., Samet, H.: Spatial join techniques. ACM Trans. Database Syst. **32**(1), 1–44 (2007). Article No. 7
11. Kalnis, P., Mamoulis, N., Bakiras, S., Li, X.: Ad-hoc distributed spatial joins on mobile devices. In: Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium (2006)
12. Kang, M.S., Ko, S.K., Koh, K., Choy, Y.C.: A parallel spatial join processing for distributed spatial databases. In: Proceedings of the 5th International Conference on Flexible Query Answering Systems, FQAS 2002, London, UK, pp. 212–225. Springer (2002). http://portal.acm.org/citation.cfm?id=645424.652610
13. Karam, O.: Optimizing distributed spatial joins using R-trees. Ph.D. thesis, Tulane University (2001)
14. Karam, O., Petry, F.: Optimizing distributed spatial joins using R-trees. In: Proceedings of the 43rd ACM Southeast Conference (2006)
15. Kwon, O., Li, K.J.: Progressive spatial join for polygon data stream. In: Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM (2011)
16. Osborn, W., Zaamout, S.: Using spatial semijoins over multiple sites in distributed spatial query processing. Can. J. Electr. Comput. Eng. **39**(2), 71–81 (2016)
17. Patel, J., DeWitt, D.: Partition based spatial-merge join. In: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, vol. 25, pp. 259–270 (1996)
18. Shekhar, S., Chawla, S.: Spatial Databases: A Tour. Prentice Hall, Upper Saddle River (2003)
19. Tan, K.L., Ooi, B., Abel, D.: Exploiting spatial indexes for semijoin-based join processing in distributed spatial databases. IEEE Trans. Knowl. Data Eng. **12**(6), 920–937 (2000)
20. Zhong, Y., Han, J., Zhang, T., Li, Z., Fang, J., Chen, G.: Towards parallel spatial query processing for big spatial data. In: Proceedings of the 26th IEEE International Parallel and Distributed Processing Symposium Workshops, pp. 2085–2094 (2012)
21. Zhou, X., Abel, D., Truffet, D.: Data partitioning for parallel spatial join processing. Geoinformatica **2**, 175–204 (1998)