

A Scaleless Data Model for Direct and Progressive Spatial Query Processing

Sai Sun, Sham Prasher and Xiaofang Zhou

School of Information Technology and Electrical Engineering
The University of Queensland, Australia
{sunsai, sham, zxf}@itee.uq.edu.au

Abstract. A progressive spatial query retrieves spatial data based on previous queries (e.g., to fetch data in a more restricted area with higher resolution). A direct query, on the other side, is defined as an isolated window query. A multi-resolution spatial database system should support both progressive queries and traditional direct queries. It is conceptually challenging to support both types of query at the same time, as direct queries favour location-based data clustering, whereas progressive queries require fragmented data clustered by resolutions. Two new scaleless data structures are proposed in this paper. Experimental results using both synthetic and real world datasets demonstrate that the query processing time based on the new multiresolution approaches is comparable and often better than multi-representation data structures for both types of queries.

1 Introduction

The driving force behind developing multi-resolution spatial database systems is to reduce the cost of spatial query processing by lowering the precision of results according to application requirements. This can lead a smaller amount of data used, thus minimizing both I/O and CPU costs of query processing. Different from conventional spatial databases which only have single resolution, a multi-resolution spatial database system is designed to support variable solutions such that the data is used at a proper resolution according to applications. Functionally, a multi-resolution system should support both direct and progressive spatial queries. A *direct query* is a spatial query whose results are not used as intermediate results for subsequent queries. This is a commonly (implicitly) assumed scenario for the traditional spatial query processing approach. The database system processes a query by only considering the query itself and uses a fixed resolution. A *progressive query*, on the other hand, considers a query as one in the context of a sequence of queries. In a spatial context this is typically a window query that is iteratively refined in both location and precision under some user's direction. An increase of precision generally accompanies a reduction in query region. Example applications include map navigations, interactive environmental analyses and spatial data mining algorithms, where users are initially not clear about

what they are looking for and require several queries to refine their search. Progressive queries are similar to those techniques that allow partial query result viewing, such as skyline queries [7]. Here, users can initiate complex operations and stop execution during processing if the results produced at that time are ‘good’ enough. The advantage is that a rough-and-ready result could take a fraction of the usual time to execute a lengthy, possibly futile, ‘complete’ query. This approach is largely targeted at mobile digital systems which have limited resources but need quick responses. Providing partial query results also avoids long response times which are unacceptable in on-line and interactive systems [3]. Multi-resolution databases are well-suited for progressive queries because they use data structures which can be retrieved either in part or in their entirety. Progressive query processing is particularly useful for spatial applications because of the volatile and diverse nature of distributed systems, and because the nature of exploratory querying is implicitly imprecise [14].

However, until now little work has been done to support multi-resolution spatial query processing. Many current multi-resolution systems still incur extraneous data retrieval because they are designed for non-progressive queries. Supporting both query types is conceptually challenging, as direct queries favour location-based data clustering whereas progressive queries require fragmented data to be clustered by resolution. In this paper, we investigate the effects of existing spatial data storage schemes for their CPU and I/O costs in the context of supporting both direct and progressive queries. Two scaleless data structures, *Position-Map* (PM) and *Bit-Map* (BM), are proposed to improve direct as well as progressive query processing performance. A cost model is proposed to estimate the storage cost of the BM scheme, and an algorithm to find the optimal base level is presented. The effect of three clustering methods for both schemes (by location, by resolution and by both) has been investigated. Our new techniques are evaluated using both synthetic and real world datasets. Experimental results demonstrate that our approaches produce comparable, and often better, performance than the traditional methods for direct queries, and significant better results for progressive queries.

The rest of the paper is organized as follows. Section 2 gives a brief introduction to related work. After analysing the advantages and disadvantages of two extreme data structures, we propose our data structures in Section 3. Section 4 discusses the progressive cost model and presents experiment results. Section 5 contains concluding remarks.

2 Background

Polygon-based structures are the most common data structure used in spatial databases. A row under this kind of scheme may be structured as in the form of *Spatial_table(ObjectId, Geometry)*, where *Geometry* is a spatial object stored as a collection of vertices. Given n spatial objects we may get n tuples. In order to minimize seek-time, objects are clustered on disk according to their spatial location. The rationale behind this is that multiple objects in close spatial prox-

imity are usually all needed together to answer queries. To further improve disk utilization data can often be compressed. The advantage of this technique is that the information of the exact geometry can be accessed directly without any extra step of object reconstruction. This technique is suitable for spatial databases in which objects are processed wholly. Because geometry elements are opaque to applications, for those operations which do not need full retention of geometry, this method causes a waste in I/O time and CPU time. Multi-resolution data structures are introduced to provide access to differently scaled representations of data, and preserve visual and topological constraints when changing scales. The multi-representation approach [16][4] pre-generates copies from the finest data at different resolution levels. In each copy, data are organized the same way as in conventional spatial database systems, i.e, based on geometry. Complex structures such as abstract cells [10] are used between different copies to measure approximation errors to maintain data coherence. This technique is, intuitively, superior in terms of data retrieval speed for single queries. However it suffers two drawbacks. First, the number of distinct resolutions made available is dictated by storage capacity. Second, progressive querying strategies cannot benefit from data replication. The total replication scheme of multi-representation defeats the purpose of progressively iterating queries as data in different resolutions are not associated, so duplicated data at a low resolution will be retrieved again when a higher resolution is required. The multi-resolution approach breaks data into fragments according to some algorithm, and then derives the proper version of the spatial data on-the-fly. It avoids the main problems of multi-representation but it costs extra CPU and I/O to derivate the data. In [6], objects are grouped into different layers according to their non-spatial attributes which are thought to be able to indicate scale. When processing a query, the non-spatial attributes are examined to check whether the corresponding objects need to be retrieved. The layered approach has drawbacks in that object must be retrieved or ignored as entirety. In [5], spatial objects are indexed under a BANG tree and stored with an explicit scale value. In a similar paper [15], objects are split into groups of vertices called vertex layers, which are also assigned a scale value and indexes as individual entities in a 3D R-tree. The two approaches allow the DBMS to deal with parts of a complex object and to approach different resolution levels by choosing different point sets. But the pre-computation of vertex layers is very complex and adding a dimension also inevitably increases the complexity of processing the index. In [9] we use z-values to encode data and adopts a Li-Openshaw algorithm for automatic map generalization. Spatial simplification can be done inside the database system. The most obvious advantage of this technique is its 'flexibility'. Each part of data may be accessed independently, thereby saving time, as there is no handling of unnecessary data. However, if a polygon has n vertexes, it will be presented as n tuples, which results in high overhead. Thus, although this method is suitable for spatial generalization, it is not good for complex spatial operations because it is difficult to compactly cluster highly fragmented data.

3 Scaleless Data Structures

In multi-resolution systems, a data object should be both fragmented and opaque. The fragmentation lets the system be scalable, which is necessary for progressive query processing, while opacity helps data clustering and reducing data reconstruction time. In order to balance these two extremes, we introduce two scaleless spatial data models in this section.

3.1 Position-Map Scheme

Under this scheme, each object is broken down into a set of levels O_1, \dots, O_m . A level contains vertexes having values within a given range, given as $[O_i^{min}, O_i^{max}]$. These ranges are disjoint (i.e., $O_i^{min} > O_{(i-1)}^{max}$ and $O_i^{max} < O_{(i+1)}^{min}$). Levels are stored with a respective delta and objected identifier. Geometry data is stored within an array object data type. As a result reconstructing an object's geometry at a certain resolution level requires that all levels of an equal or lesser delta value be joined. Thus $O_m = O_1 \oplus O_2 \oplus \dots \oplus O_m$. To ensure correctness of reconstruction, the position of each vertex in O must be recorded explicitly. A simple way to achieve this is by incorporating vertex positions into the geometry array. A position map is used to store information regarding the position of points in a level. Bits corresponding to those stored within the representation are flagged thus requiring a single scan of the position map during reconstruction; the same complexity needed when explicitly storing point positions. For example, for an object of 10 vertexes with its 2th, 5th and 7th in level i , the data in O_i should be $(x_2, y_2), (x_5, y_5)$ and (x_7, y_7) , and PM should be the bit sequence '0100101000'.

Because several levels may be needed together to construct an object, levels are clustered together according to their associated `objectID`. Compared with a traditional single-resolution spatial data structure, a small overhead is incurred to split an object over several tuples. This scheme may, however, save data retrieval time for progressive queries, because only the data with high resolutions than that in previous queries need to be fetched. It must be noted that partial replication exists in this scheme. Next, we introduce the BM Scheme.

3.2 Bit-Map Scheme

Points can be encoded using z-values [11], such that the resolution of a point data (represented by a rectangle surrounding the point) can be reduced by truncating the right-most n bits of its z-values, where n is the resolution used for initial simplification [8]. At high resolutions an object will be represented by more z-values than at a low resolution. Hence some of these values will share a common prefix. Put another way, every point at a low resolution will exist in every higher resolution representation. By truncating z-values we can obtain a prefix to reduce their precision. However, under the PM scheme, points from lower resolution levels may be needed to construct a high-resolution object. Therefore some extra information must be stored so that z-valued can be reconstructed to the precision required by the user from a starting prefix (resolution).

An object is reconstructed by first selecting a base resolution from the dataset, at which all points are stored. Points are then truncated to the resolution length of this resolution and stored in a point array with duplicate points (after z-value truncation) removed. We select the resolution at least at which the total number of unique vertices ensures each object in the dataset has at least c points, where $c \geq 1$ (typically $c = 4$); when $c < 1$, the base levels of two objects may be equivalent, hence partially redundant. Second, for each subsequent resolution level a *bitmap* is constructed to map the extension of ancestors into descendants. Because the degree of extension is not known, we must also indicate which ancestors an extension applies to. As illustrated in figure 1, 3 bits are used to denote a single extension: the first two bits for the ordinate, and the third bit to indicate whether the mapping applies to the next ancestor point (0 - continue; 1 - next point). Each is stored with a resolution value. The original object is reconstructed by iteratively extending point levels using a bitmap of the next higher resolution.

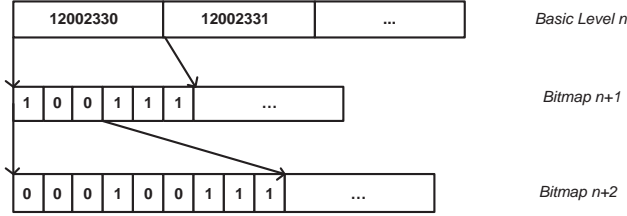


Fig. 1. Base level and bitmaps

In Figure1, for base level n , we have a point '12002330' and it expands to two new points in level $n + 1$ ('120023302' and '120023303'). '120023302' further expands to three new points in level $n + 2$. They are '1200233020', '1200233022' and '1200233023'. Note that if the change in the number of points between resolutions is small, multiple bitmaps can be compressed into a single bitmap. In this case, for k bitmaps compressed each extended point will require $2k + 1$ bits.

3.3 Storage Cost Model

We now provide a basic storage cost model for Bitmap-based scheme. Let n_r denote the number of all points whose resolution is less than or equal to r . Then, a sequence $\{n_0, n_1, \dots, n_i, \dots, n_m\}$, ($0 \leq i < m$), where m is the highest resolution level of the dataset, can be used to describe data distribution of the number of points in a different resolution. The resolution of the base level that stores a sequence of truncated z-values of vertices is denoted as i . Each z-value requires a bits of storage, i.e. $a = 2$. Finally the number of points within a certain resolution range $[r, r + 1]$ is denoted as $n_r = n_{r+1} - n_r$. Note n_r is never

less than 0. Given this, we may determine that the storage of the traditional single-resolution scheme as:

$$S_o = n_m \times m \times a = (n_i + \sum_{r=i}^{m-1} \Delta n_r) \times m \times a \quad (1)$$

And, the storage of the bitmap method is:

$$S_{bit} = n_i \times i \times a + \sum_{r=i+1}^m n_r \times 3 = n_i \times i \times a + n_i \times (m-i) \times 3 + \sum_{r=i}^{m-1} (m-r) \times \Delta n_r \times 3 \quad (2)$$

When substituting S_o into equation 2, we have:

$$S_{bit} = S_o - n_i \times (m-i) \times (a-3) - m \times (a-3) \times \sum_{r=1}^{m-1} \Delta n_r - 3 \times \sum_{r=1}^{m-1} \Delta n_r \times r \quad (3)$$

We can see that storage of Bitmap method is dependant on i , m and n_r . Because m is determined by the precision of data set and n_r is decided by data distribution, the actual storage for S_{bit} is affected by the selection of i . Statistical analyses reveal that the increase of Δn_r satisfies some rules which is small on both ends and large in the middle, forming a bell curve [2], which can be approximated by a Gaussian distribution. Figure2(a) shows the curves of the data distributions for the synthetic and SEQUOIA datasets. We can regard the curve as a probability density function of a variable R , where $\frac{\Delta r \Delta n_r}{N}$ is the probability that R is located in the range $[r, r + \Delta r]$. Following this, the Gaussian distribution approximation can be given as follows:

$$\Delta n_r = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(r-\mu)^2}{2\sigma^2}}, \mu = \frac{\sum_{r=1}^{m-1} \Delta n_r \times r}{\sum_{r=1}^{m-1} \Delta n_r}, \sigma = \sqrt{\frac{\sum_{r=1}^{m-1} (r-\mu)^2}{m-1}} \quad (4)$$

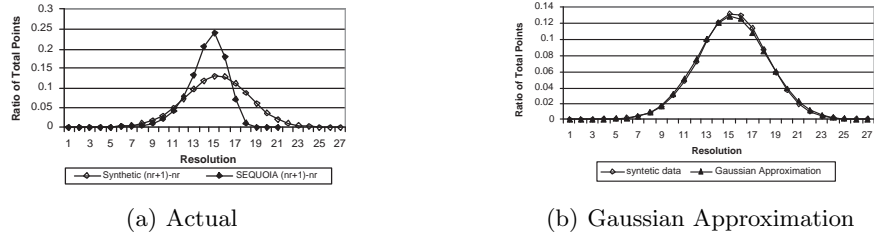


Fig. 2. Data Distributions

Thus μ is close to the level value that has the biggest Δn_r and σ is the standard deviation of the Gaussian distribution. Figure2(b) shows that the Gaussian

approximation is a very good indication of the actual distribution of the synthetic dataset.

For simplification, we set $a = 3$ in equation 3, then:

$$S_{bit} = S_o - 3 \times \sum_{r=i}^{m-1} \Delta n_r \times r = S_o - 3 \times \sum_{r=i}^{m-1} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(r-\mu)^2}{2\sigma^2}} \times r \quad (5)$$

$$\begin{aligned} & S_o - 3 \times \frac{\sigma}{\sqrt{2\pi}} \left(e^{-\frac{(m-1-\mu)^2}{2\sigma^2}} - e^{-\frac{(i-\mu)^2}{2\sigma^2}} \right), \quad (i \leq m-1 \leq \mu) \\ & = S_o - 3 \times \frac{\sigma}{\sqrt{2\pi}} \left(2 - e^{-\frac{(m-1-\mu)^2}{2\sigma^2}} - e^{-\frac{(i-\mu)^2}{2\sigma^2}} \right), \quad (i \leq \mu \leq m-1) \\ & S_o - 3 \times \frac{\sigma}{\sqrt{2\pi}} \left(e^{-\frac{(i-\mu)^2}{2\sigma^2}} - e^{-\frac{(m-1-\mu)^2}{2\sigma^2}} \right) \quad (\mu \leq i \leq m-1) \end{aligned} \quad (6)$$

We can see that as i decreases, less storage is needed, which generally will reduce disk I/O costs. Consequently, the time complexity of reconstruction is $O(\sum_{r=i}^{m-1} n_r)$. So a small i incurs a larger cost for object reconstruction. Thus, if i is too small the time used to reconstruct data may negate any performance advantage caused by less storage. To different systems, applications can find a trade-off point for i between this two factors according to above discussion. In our experiment for synthetic data, we chose $i = 10$, $m = 28$. If $a = 2$, the reduction of storage is 24,270,821 bits; if $a = 3$, the reduction is 63,508,107 bits. For the real data, we chose $i = 15$, $m = 22$. If $a = 2$, the reduction is 22,844,603 bits and if $a = 3$, the reduction is 59,481,354 bits.

In order to further improve the storage efficiency, we combine and compress the bit map of high resolution levels with small. Since at the highest several resolution levels, a small increase of the number of points will weaken the benefit of bit map and enhance the cost of reconstruction. An extreme case is that when level k and level $k+1$ contain the same number of points. Here each point in the bit map for level $k+1$ is associated with a flag bit that always set to be 0, which is a waste. If we can combine and compress the bit map of multiple continuous levels, say 3 levels, we can directly derive z-values of points for level $k+3$ from data of level k . Therefore each point in the bit map will contain 6 bits (for 3 levels) plus one flag bit.

4 Experimental Evaluation

In this section, we give a simple progressive query cost model first. Then we study the effect of different data clustering approaches to the various storage structures discussed thus far.

4.1 Progressive Query Cost Model

We defined a progressive query set Q as a series of queries in numerical order as $\langle Q_1, Q_2, \dots, Q_n \rangle$. For Q_i , we use l_i to represent its resolution level, $S(Q_i)$ to represent the set of Object IDs in its query result, and A_i to represent its

query area. Therefore, those parameters satisfy the following rules: $l_1 \leq l_2 \leq \dots \leq l_n$, $S(Q_1) \supseteq S(Q_2) \supseteq \dots \supseteq S(Q_n)$, $A_1 \supseteq A_2 \supseteq \dots \supseteq A_n$. Generally, the time cost $T(Q)$ for certain spatial queries is mainly composed of two parts: Data retrieval $D(Q)$ and Query processing $J(Q)$. The cost for data retrieval is primarily composed of I/O whilst for join processing more time is spent on CPU calculation. Hence, we consider the time needed for progressive query Q as: $T(Q) = T(Q_1) + T(Q_2) + \dots + T(Q_n)$, where $T(Q_i) = D(Q_i) + J(Q_i)$, $i = 1 \dots n$. Since queries are normally specified with a certain resolution level l and area A , we may incorporate l , A into $D(Q)$ and $J(Q)$, giving $D(Q, l, A)$ and $J(Q, l, A)$ respectively. Thus, for a multi-representation spatial database, the time cost of a progressive query Q would be:

$$T_M = D(Q_1, l_1, A_1) + J(Q_1, l_1, A_1) + \dots + D(Q_n, l_n, A_n) + J(Q_n, l_n, A_n) \quad (7)$$

Here, $l_1 \leq l_2 \leq \dots \leq l_n$ and $A_1 \supseteq A_2 \supseteq \dots \supseteq A_n$

For a conventional spatial database with only a fixed resolution l_{fix} , in order to get the same query result, l_{fix} should be not less than l_n , and T would be:

$$T_S = D(Q, l_{fix}, A_1) + J(Q, l_{fix}, A_1) \quad (8)$$

Comparing equation 7 and 8, we can see that T_M is obvious less than T_S when $l_n \ll l_{fix}$, which is the case that users stop query according to partial query result. Two factors influence T_M greatly. First is the decrease in A_i . A larger reduction means better performance. In a worst case scenario $A_1 = A_2 = \dots = A_n$. Second is the redundancy between the data of different resolution levels. Too much redundancy will result in noticeable overhead in accessing data. That is the reason why multi-representation is not suitable for progressive queries.

4.2 Performance Study

In the following sections we investigate the effect of four different data organization methods: single resolution (SR), multi-representation (MR), Position Map (PM) and Bit Map (BM), on performance over various aspects of the multi-resolution spatial join on a window. Window query is a basic geometric selection because spatial operations frequently involve location as a condition [1]. Spatial join is the most commonly used way to combine information from two or more spatial relations [13]. For MR all objects of a particular representation are clustered together. Objects are ordered according to a z-curve on the centroid of their MBRs. The same organization is used for objects in SR. For PM and BM, spatial data are clustered 1) based on location 2) based on resolution and 3) based on location then resolution (implemented by 3D z-value), termed PM_O, PM_R, and PM_3D; BM_O, BM_R, and BM_3D respectively. The aim of our tests is to determine the most efficient multi-resolution scheme overall for both direct and progressive queries. Performance is measured in both number of I/O accesses and time. While I/O is generally considered a more important factor, if seek time is high then overall response time may be higher than expected. We aim to minimize I/O cost while maintaining a low processing time.

Environment and Data. All tests were conducted in an Oracle database and JDK 1.3 environment on a PIII-800 256Mb system. Point arrays are stored using a VARRAY data structure of NUMBER types. A RAW binary data type is used to store each bitmaps. Disk pages are 4kb. Every point array and bitmap is associated with an object identifier and resolution value, r , in the same tuple. Standard B^+ Trees are placed on these fields. We used the California SEQUOIA polygon dataset [12] for join testing. The data was split into two layers of regular feature polygons, and polygon holes. Very large and very small objects were removed leaving 20,137 objects in layer A, and 5,798 in layer B (holes). In final dataset contains 2,635,065 of the original 3,207,350 data points; i.e. removed objects were treated as outliers. All layers are supplemented with an auxiliary table that stores MBR information for each object. Because the same MBR table is used for different schemes we do not elaborate on performance of the filter step.

Query Process. The join query on a window $w = \{x_{min}, y_{min}, x_{max}, y_{max}\}$ for a resolution r can be informally given as: ‘Select objects from layer A and layer B that are inside w and that intersect at resolution r ’. Let an object O , stored in p resolution-specific fragments, be the combination of all vertices stored in those fragments, or $O = \{O_1 \oplus \dots \oplus O_p\}$ where $1 \leq r \leq p$. In a direct query the candidate set produced by the filter step, S , for some arbitrary window, is a set objects obtained as the joining of their respective r fragments. Every fragment of an object in S is retrieved contiguously before moving on to the next candidate. A progressive query is issued over a series of windows $\hat{W} = \{w_1, \dots, w_n\}$ where $w_1 > \dots > w_r$, corresponding to a series of result sets $\hat{S} = \{S_1, \dots, S_r\}$ where $|S_1| \geq \dots \geq |S_r|$. For each set S_k in \hat{S} a single resolution-specific fragment O_k of each object is retrieved. After the first set is processed an arbitrary object is described as $O' = \{O_k\}$ where $k = 1$. Each set is processed iteratively, and new fragments are added to old fragments, until S_r where $O' = \{O_1 \oplus \dots \oplus O_r\}$.

For reference the filter step comprises of a hash-join on the MBRs of objects from both layers within w and produces an unordered candidate set. Ordering this set is identical across all tested methods, hence we do not elaborate on it in the test results. We measure retrieval time as the total time required to fetch candidate object geometries, and bitmaps where applicable, from disk and perform object reconstruction.

Test Results. We compare the single-resolution and multi-resolution schemes under direct and progressive queries. In progressive querying an initial 100% window on the data space is issued at resolution 15, which gives data with a precision of 128m. The window is eventually refined to a 0.39% window with data at resolution 22, with 1m precision. The results of the direct query test (figure 3) show comparable performance of all schemes for smaller window sizes. We separate the variant approaches for the PM and BM schemes for easy visualization. The 3D variations, shown to perform best, are then compared with the other schemes. Since data loads are relative small this is expected. The PM_O variation has an overall worst performance because it does not consider resolution whereas both 3D clustering schemes prove more efficient than their lower-dimensional counter-

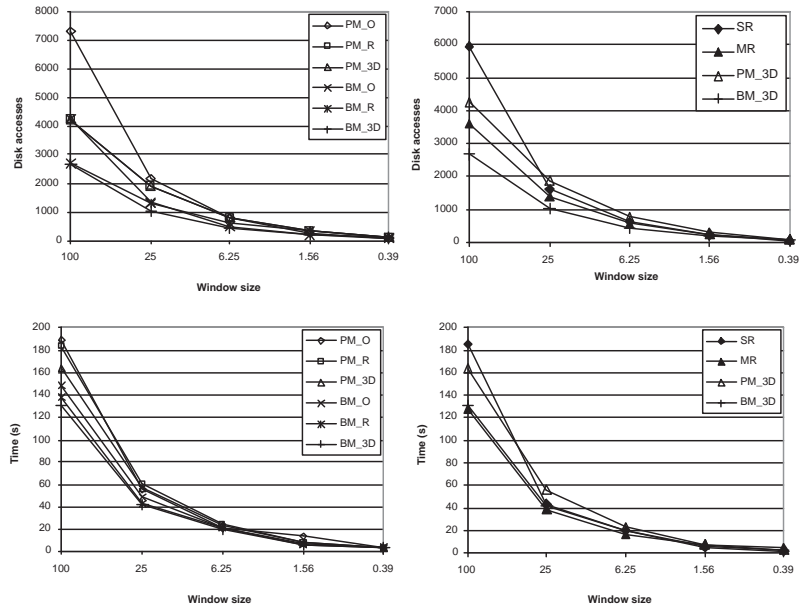


Fig. 3. Direct query test

parts. PM also exhibits a larger time and I/O requirement than bitmap schemes, which can be attributed to its higher storage overhead. Expectedly, MR has an ideal time performance, however incurs higher data retrieval than our compressed bitmaps. Note I/O reduction does not correlate directly to retrieval time due to the inherent variation in seek time with different clustering methods. At resolution 15 the multi-resolution schemes require 47% of the original data load. This figure is close to the I/O retrieval in BM_R and BM_3D schemes (45-44% of the original), though is noticeably lower than that seen in the MR (60%) and all PM variations (minimum 71%). Note the bitmap scheme has an I/O reduction slightly lower than the 47% reduction because compression in the bitmaps reduces overall storage by 30%. To illustrate storage requirements, under SR the data table size is 28.6Mb, 121.6Mb under MR, 38.9Mb in PM, and 19.9Mb in BM.

Figure 4 shows the total performance time and I/O costs for the progressive query calculated at each step of five iterations. The results show a clearer distinction between the single-resolution and multi-resolution methods. The bitmap approach performed comparably with MR while maintaining an overall lower I/O cost. We also note that the I/O cost of the bitmap schemes was roughly 50% that of PM. Cumulatively this produces a lower total cost making it scalable to larger operations. The data loads processed under each window is given in figure 5. The bitmap compression technique manages to save approximately one third

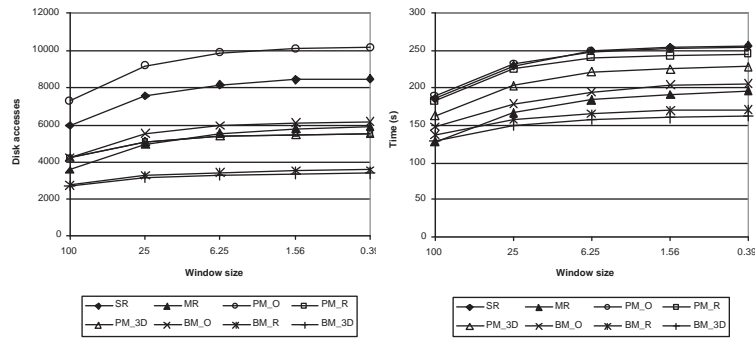


Fig. 4. Progressive query test

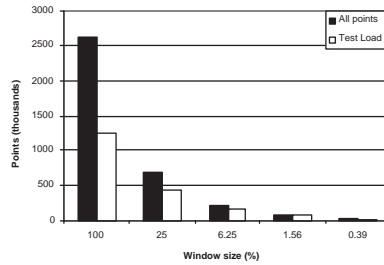


Fig. 5. Data Loads

in storage. PM uses the same amount more than a single-resolution scheme. If we consider that each x,y coordinate requires an index value, a roughly 33% increase in size is justified.

5 Conclusions

In order to support direct queries and progressive queries at the same time, data should be both fragmented and opaque in multi-resolution systems. In this paper, we investigated the advantages and disadvantages of the common data structure used in current databases and the fragment data structure used in our previous work. To balance the two extreme data structures, we introduced two scaleless data structures - the Position-Map and the Bit-Map. Between them, the Bit-Map scheme is more efficient because this technique removes the replication existing in most current schemes. To estimate the storage of the Bit-Map scheme, we constructed a mathematical model and analysed how to find the optimal value for base level in Bit-Map scheme. Then, we discuss the cost model for progressive query. The aim of our tests is to determine the most efficient multi-resolution scheme overall for both direct and progressive queries. Our experiment

results demonstrated that our approaches have comparable, often better, performance than traditional methods to direct queries, and superior performance under progressive queries.

Acknowledgment: The work reported in this paper has been partially supported by grant DP0345710 from the Australian Research Council.

References

1. A. Aboulnga and J. R. F. Naughton. Estimation of the cost of spatial selections. In *ICDE*, pages 123–134, 2000.
2. P. J. Diggle. *Statistical analysis of spatial point patterns*. Oxford University Press, 2003.
3. J. Dittrich, B. Seeger, D. S. Taylor, and P. Widmayer. Progressive merge join: A generic and non-blocking sort-based join algorithm. In *VLDB*, pages 299–310, 2002.
4. A. U. Frank and S. Timpf. Multiple representations for cartographical objects in a multi-scale tree - an intelligent graphical zoom. *Computers and Graphics*, 18(6):823–829, 1994.
5. M. Horhammer and M. Freeston. Spatial indexing with a scale dimension. In *SSD*, pages 52–71, 1999.
6. R. Kanth, D. Agrawal, A. E. Abbadi, A. K. Singh, and T. R. Smith. Parallelizing multidimensional index structures. In *IEEE Symposium on Parallel and Distributed Processing*, pages 376–383, 1996.
7. D. Kossmann and F. Ramsak and S. Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *VLDB*, pages 275–286, 2002.
8. P. Prasher. Perfect cartographic generalisation and visualisation. In *VDB*, 2002.
9. S. Prasher, X. Zhou, and M. Kitsuregawa. Dynamic multi-resolution spatial object derivation for mobile and WWW applications. *J. WWW*, 6(3):305–325, 2003.
10. E. Puppo and G. Dettori. Towards a formal model for multiresolution spatial maps. In *SSD*, pages 152–169, 1995.
11. H. Samet. *Applications of Spatial Data Structures*. Addison-Wesley, 1990.
12. M. Stonebraker. An overview of the sequoia 2000 project. *Digital Technical Journal of Digital Equipment Corporation*, 7(3):39–49, 1995.
13. C. Sun, D. Agrawal, and A. E. Abbadi. Selectivity estimation for spatial joins with geometric selections. In *EDBT*, pages 609–626, 2002.
14. R. Vijayshankar. Partial results for online query processing. In *SIGMOD*, pages 275–286, 2002.
15. S. Zhou and C. B. Jones. Design and implementation of multi-scale databases. In *SSTD*, pages 365–384, 2001.
16. S. Zhou and C. B. Jones. A multi-representation spatial data model. In *SSTD*, pages 33–51, 2003.