2006

Clustering Spatial Data for Join Operations Using Match-based Partition

Jitian Xiao
Edith Cowan University

10.1109/CIMCA.2005.1631513
This conference paper was originally published as: Xiao, J. (2006). Clustering Spatial Data for Join Operations Using Match-based Partition. Proceedings of International Conference on Computational Intelligence for Modelling, Control and Automation. (pp. 471-476). Vienna, Austria. IEEE Computer Society Press. Original article available here

Clustering Spatial Data for Join Operations Using Match-based Partition

Jitian Xiao
School of Computer and Information Science, Edith Cowan University,
2 Bradford Street, Mount Lawley, WA 6050, Australia
E-mail: j.xiao@ecu.edu.au

Abstract

The spatial join is an operation that combines two sets of spatial data by their spatial relationships. The cost of spatial join could be very high due to the large sizes of spatial objects and the computation-intensive spatial operations. In spatial join processing, a common method to minimize the I/O cost is to partition the spatial objects into clusters and then schedule the processing of the clusters such that the number of times the same objects to be fetched into memory can be minimized. In this paper, we propose a match-based approach to partition a large spatial data set into clusters, which is computed based on the maximal match on the spatial join graph. Simulations have been conducted and the results have shown that, when comparing to existing approaches, our new method can significantly reduce the number of clusters produced in spatial join processing.

1. Introduction

The spatial join is a common spatial query type that requires a high processing cost due to the large volume of spatial data and the computation-intensive spatial operations. Spatial join queries usually access a large number of spatial data.

To reduce the CPU and I/O costs for spatial join processing, most spatial join processing methods are performed in two steps (i.e., *filter-and-refine* approach). The first step chooses pairs of data that are likely to satisfy the join predicate. The second step examines the predicate satisfaction for all those pairs of data passing through the filtering step.

During the filtering step, a conservative approximation of each spatial object is used to eliminate objects that cannot contribute to the join result, and a *weaker* condition for the spatial predicate is applied on the approximations. This step produces a list of *candidates* that is a superset of the joinable candidates. These candidates are usually represented as pairs of object identifiers. All candidates are then checked in the refinement step by applying the spatial operation on the full descriptions of the spatial objects to eliminate the "false drops". The join cost can be reduced because the weaker condition is usually computationally less expensive to evaluate and the approximations are small in size than the full geometry of spatial objects.

The filtering algorithms were well studied [2, 3, 4]. However, using the same weaker condition, different filtering algorithms will produce candidates in different orders. Such differences can influence significantly on the refinement cost [4]. It is necessary to cluster the candidate set of the filtering result in order to reduce the I/O cost of the refinement step [3].

Let S and T be the two spatial database tables for spatial join operation, denoted by $S \chi T$. Objects in S and T are indexed by their unique IDs. The spatial data of these objects can have different sizes, i.e., they are non-uniform sized. The filter operation of the spatial join produces a set of pairs of S and T objects. Let F be the set of ID pairs produced by the filter operation:

$F = \{(sid, tid) |$ *sid* and *tid* are IDs of objects in S and T, respectively, that meet the weaker join condition$\}$

where an ID pair $(sid, tid) \in F$ is called a *candidate*. Figure 1 (a) shows an example of F. Note that F is available in the main memory after the filter operation. F contains only IDs of the candidates, not the data objects.

The refinement step is to perform $S \chi T$ on the pairs of objects indexed by F to produce the final join results. At this step, the S and T objects need to be fetched into the main memory for the full spatial join test. Since some candidates may have join operation with several others, it needs to be fetched several times into the memory for the join operations. Taking the example of Figure 1 (a), B1 has join relation with A1, A2 and A5. After the join operation with A1 and A2, it may need to be fetched into the memory again when it joins with A5, if it was flushed out of the memory.

Proceedings of the 2005 International Conference on Computational Intelligence for Modelling, Control and Automation, and International Confe
Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'05)
0-7695-2504-0/05 $20.00 © 2005 IEEE

Our method is to cluster the objects into groups and then fetch objects in the same group into the memory for processing in a batch. The number of times an object to be fetched can thus be reduced.

It is very important to reduce the I/O cost of fetching the full geometry of spatial objects, because it contributes a significant portion of the total cost of performing a spatial join operation. To consider the I/O cost, we take the spatial object size into account. The spatial object sizes can differ greatly from one to another. For example, while a spatial point object occupies only several bytes of storage, a large polygon object in a road map may have up to tens of thousands of edges that occupy several megabytes of storage. The I/O cost, in this paper, is measured in terms of the size of spatial data that are fetched into the memory for the refinement operation

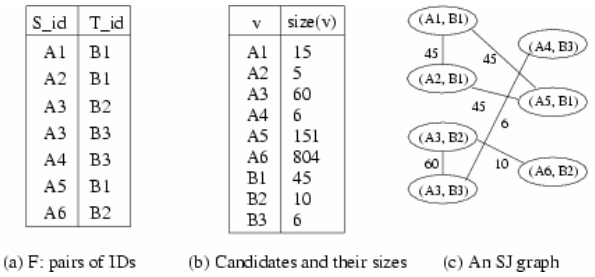

Figure 1. An example of a candidate set and SJ graph.

The rest of the paper is organised as follows: Section 2 describes a graph model to characterize the spatial data clustering problem. Section 3 proposes a match based method to partition spatial objects into clusters. Section 4 analyses the complexity of the proposed algorithm. Section 5 presents our simulation results, and Section 6 concludes the paper.

2. Clustering spatial join operations over graph model

For a given candidate set *F*, we introduce a weighted graph G_F =(V, E, w), called *Spatial Join* (SJ) graph, to represent the join relationships between spatial objects referenced in *F*. Intuitively, the node set V contains all pairs of IDs in *F*, i.e., each node corresponds to a join operation between a pair of objects. The edge in the SJ graph G_F is to reflect the overlapping status of the objects between different join operations represented by the nodes. Formally, V = $\{(v_1, v_2) \mid \{(v_1, v_2) \in F\}$, and E = $\{((v_1, v_2), (u_1, u_2)) \mid \{(v_1, v_2), (u_1, u_2) \in F, \{v_1, v_2\} \cap \{u_1, u_2\} \neq \emptyset\}$. The weight of an edge e = $((v_1, v_2), (u_1, u_2))$ is defined as the size of the object overlapped between $\{v_1, v_2\}$ and $\{u_1, u_2\}$, i.e., w(e) = size(x) if $\{x\} = \{v_1, v_2\} \cap \{u_1, u_2\}$. Note that $|\{v_1, v_2\} \cap \{u_1, u_2\}| \leq 1$ since different pairs of IDs in *F* are distinct. For example, the object sizes of the candidates in *F* of Figure 1 (a) are given in Figure 1 (b). Figure 1 (c) shows the corresponding SJ graph with weights shown beside edges.

Let V_C be the set of candidates referenced in F, i.e.,$V_C = \{x \mid \exists v: (x, v) \in V \text{ or } (v, x) \in V\}$. The total I/O cost, $C_{I/O}$, of fetching objects referenced in *F* varies in the range of

$$\sum_{x \in V_c} size(x) \leq C_{I/O} \leq \sum_{(v_1, v_2) \in V} (size(v_1) + size(v_2))$$

The lower bound is achieved when each object is fetched only once, and the upper bound is met where an object is fetched every time when it is referenced.

To reduce $C_{I/O}$, our strategy is to group objects referenced in *F* into clusters such that objects inside a cluster are more likely to join with each other than those across clusters. The objects in the same cluster are brought into the memory together and processed in a batch. To achieve this, we partition V into *m* subsets $V_1, V_2, \ldots, V_m$ such that $V_i \cup V_j = V$ and $V_i \cap V_j \neq \emptyset$ for $i \neq j$. Let $E_i = E \cap \{((v_1, v_2), (u_1, u_2)) \mid \{(v_1, v_2), (u_1, u_2) \in V_i\}$, i=1, 2, …, *m*. Then $G_i = (V_i, E_i, w|_{Ei})$ is a subgraph of G_F. Let *q* be the maximum memory capacity that is used to hold spatial objects for refinement operation. For any G_i, $1 \leq i \leq m$, the total size of objects in G_i must be less than or equal to *q*, i.e.,

$$\sum_{v \in c_i} size(v) \leq q \tag{1}$$

where $c_i = \{x \mid \exists v: (x, v) \in V_i \text{ or } (v, x) \in V_i\}$ is the set of objects referenced in V_i, called the i_{th} cluster. We assume that the total size of each pair of objects referenced in *F* must be less than or equal to *q*; otherwise no solution exists.

The objects in a cluster are fetched together into memory for processing, and are thrown away when the next cluster is processed. Therefore, the total I/O cost can be expressed as

$$C_{I/O} = \sum_{v \in V_C} |\{j \mid v \in c_j\}| \cdot size(v) \tag{2}$$

or

$$C_{I/O} = \sum_{v \in V_C} size(v) + \sum_{v \in V_C} (|\{j \mid v \in c_j\}| - 1) \cdot size(v) \tag{3}$$

where $|\{j \mid v \in c_j\}|$ denotes the total number of clusters that contains object *v*. Let y be

$$y = \sum_{v \in V_C} (|\{j \mid v \in V_j\}| - 1) \cdot size(v) \tag{4}$$

y is the total object size that may need to be reloaded for processing objects that appears in more than one cluster, when processing different clusters. If each

object belongs to no more than three clusters, then y can be simplified as

$$y = \sum_{v \in c_i \cap c_j, i \neq j} size(v) \quad (5)$$

Since $\sum_{v \in V_c} size(v)$ is a constant, the optimization of $C_{I/O}$ is to find a partition of G_F such that y is minimized. The problem of minimizing y in graph G_F is *NP* complete [5].

3. The match based algorithms for spatial data partitioning

In [1], a matrix-based algorithm was proposed to cluster non-uniformed spatial objects referenced in F such that the object in the same cluster are closely related (i.e., they are more likely to join with each other). However, when the number of objects becomes larger (say thousands of objects), the space requirement of the matrix-based algorithm increases dramatically. This suggests alternative algorithms that are applicable to cases where a large number of spatial objects are involved in a spatial join.

To ease the discussion, we introduce some terms: Two partitions, in a given SJ graph, are said to be *adjacent* (*connected*) if some of their nodes are adjacent (connected) in the SJ graph. The *size* of a partition refers to as the total size of objects referenced by nodes in the partition. The *weight (of edges)* between two partitions is the sum of the weights of edges that connect nodes between the two partitions.

For a given SJ graph $G_F = (V, E, w)$, where $V = \{v_1, v_2, \ldots, v_k\}$[1], we partition V by initially taking each node of V as a partition, and then continually combining two (or more) partitions to form larger ones. The partitioning of V is commonly represented by a partitioning vector P of length k, such that for every node $v \in V$, $P[v]$ is an integer between 1 and m, indicating the partition to which v belongs. So combining partitions V_i and V_j (into V_i) can be easily implemented as: *for (l=1; l<=k; l++) {if (P[l] == j) P[l] = i;}*.

To minimize y in (4), we consider two partitioning criteria, i.e., *local* and *global*, when selecting partitions to combine. The local criterion aims to select partitions whose combination would cause a greater reduction of y. Consider two partitions V_i and V_j. Let $W_{i\sim j}$ be the weight of edges between V_i and V_j. If we combine V_i and V_j to form a single partition, then the total reduction of y will be $W_{i\sim j}$. Hence, by selecting a pair of partitions that have a greater weight between them, we can decrease y by a greater amount.

The global criterion takes into account the global effects when combining partitions. Consider two partitions V_j and V_l, both connecting to V_i by edges of same total weight. Let A_i^j (and A_i^l, respectively) be the set of adjacent partitions of V_j (and V_l, respectively) that connect V_i, and E_i^j (and E_i^l, respectively) the total weight of edges between V_j and A_i^j (and A_i^l., respectively). If $E_i^j > E_i^l$, we combine V_i and V_j, otherwise V_i and V_l, to form a new partition, say V'. In this way, we get a greater weight of edges between V' and its adjacent partitions. This increase the chance to combine partitions of higher weight, thus increase the reduction of y in the following combinations.

To meet both the local and global criteria we propose a maximal match based partition method to cluster objects over the SJ graph. A *match* of a graph is a set of edges; any two of them are not incident to the same node. A *weighted matching* (WM) problem is, for a given graph G, to find a match of G such that the sum of the edge weights of the match is maximal. The WM problem was solved by J. Edmonds [5] and the complexity of his algorithm is $O(n^3)$, where n is the number of nodes of G. The resultant match is called a *maximal match*[2] of the graph.

Our partitioning method works in the following way: Initially, we have $m = k$ partitions, each contain one node (of the SJ graph). Then, we reduce m by combining the existing partitions using maximal match. We employ Edmonds' algorithm to produce a maximal match from $G_0=G_F$, then combine pairwisely the matched nodes (or partitions) to form larger partitions. At this point, we construct a coarsened graph G_1 which has those newly formed partitions as its nodes (unmatched nodes of G_0 are copied over to G_1). The edges between the nodes of G_1 are defined based on their edges in G_0, as follows: For any pair of nodes (partitions), there is an edge between them. If the two partitions are adjacent (i.e., some of their nodes are adjacent in G_0), and the total size of the objects referenced in the two partitions is less than or equal to q (i.e., the maximum memory capacity as defined in formula (1)), the weight between them is set as the weight between the two partitions, otherwise the weight is set as 0, suggesting that the two partitions will not be combined in the following iterations[3]. At

[1] By definition, each node v_i of an SJ graph is in the form of (u_i^1, u_i^2), i.e., a pair of IDs referenced in F.

[2] A match is *maximal* if any edge in the graph that is not in the match has at least one of its endpoints matched, and the sum of the edge weights of the match is maximal among all matches of the graph.

[3] Normally, two nodes with a weight of 0 will not be matched in a maximal match because such a match does not contribute to the total weight of the match. However we

this point, next round of iteration begins to find a maximal match from G_1. This match-and-combination process continues until no more matches can be found or no partitions can be expanded (e.g., combination of any two partitions would violate formula (1)). The algorithm can be described as below:

Algorithm *matchBasedClustering*(*G*)
Input: G = (V, E, w); // $V = \{v_1, v_2, , ..., v_k\}$.
Output: $c_1, c_2, , ..., c_m$; // clusters of objects referenced in *F*.
Begin
[1] Find a maximal match *M* of *G* using *Edmonds*' algorithm;
[2] ***if*** no match was found ***or*** no pair of matched nodes can be combined
[3] ***then*** compute clusters $c_1, c_2, ..., c_m$ from V*;*
return$(c_1, c_2, ..., c_m)$;
[4] ***for*** all matched nodes // Coarsen *G* by collapsing // matched nodes of *M* to produce a coarser graph *G*'
[5] ***if*** v_i and v_j are a pair of matched nodes, and

$$\sum_{x \in c_i} size(x) + \sum_{x \in c_j} size(x) - w(v_i, v_j) \le q$$

[6] ***then*** combine v_i and v_j to from a node of *G'* ;
[7] copy matched but uncombined nodes over to *G'*;
[8] copy unmatched nodes over to *G'*;
[9] update edges and the weights (of edges) between the newly formed nodes of *G'*;
[10] *matchBasedClustering*(*G*');
[11] ***return***;
End

Example 1: To illustrate how the algorithm works, we explore a part of the execution of the algorithm. Let *q*=1600. Figure 2 (a) shows a coarsened graph after some rounds of recursive execution of the algorithm, in which each node represents a partition of the original SJ graph (not presented here). The size of the spatial objects referenced in a node (partition) is shown beside the node (i.e., in a square bracket). The number beside an edge is the weight between the two nodes. For simplicity, all edges with a weight 0 were not given in the graph. In the next round of execution, the first step produces a maximal match *M*, which has 7 edges, with a total weight of 875, as shown by thick bold edges in the figure 2 (a). There is only one unmatched node (i.e., node labelled by 11) in the graph. Then pairs of matched nodes (partitions) are combined to form single nodes of the next level coarsened graph, as shown in figure 2 (b). Labels of the matched nodes are put together, separated by "/", as the label of the formed nodes in the coarsened graph (e.g., label "1/4" in Figure 2 (b) indicates that the node was formed by combining nodes label by 1 and 4 in the figure 2 (a)). It shows that 7 nodes of the graph were formed from the matched nodes of Figure 2 (a), and only one node was copied from the unmatched node. By this matching-and-combination process, the total edge weight was reduced by 875 (out of 1490).

Similarly, the next round of execution of the algorithm results in a coarsened graph as shown in figure 2 (c). During the edge weight update step (i.e., in step [6]), the edge weight between the nodes labelled 5/9/6/10 and one labelled 12/13/14/15 were set as 0 (although the total size of objects overlapped between them is 60) because their combination would results in violation of formula (1). In this way, we force the algorithm not to match them in the next round execution of the algorithm if there is any other choice. The case between the nodes labelled by 7/8/10 and 12/13/14/15 is similar, as shown in dotted lines. The last round of recursive execution of the algorithm produces a maximal match containing two edges, of which only one leads to a combination of the linked partitions. The other pair of nodes was not combined because the restriction enforced by the rule of formula (1). The algorithm output three data clusters: The first cluster contains all spatial IDs that were originally referenced by the nodes (partitions) labelled by 1, 2, 3, 4, 5, 6, 9 and 10 in figure 2 (a), of total object size 1576. The second contains all spatial IDs that were originally referenced by the nodes (partitions) labelled by 7, 8 and 11, of total object size 470. And the last one contains those originally referenced by the nodes (partitions) labelled by 12, 13, 14 and 15. This cluster has a total size of 1578. The sizes of overlapping objects between these clusters are 60, 60 and 30, respectively, as shown in brackets beside edges in Figure 2 (d).

4. Algorithm analysis

We now analyse the complexity of the algorithm *matchBasedClustering*. For a given SJ graph with *n* nodes, line 1 needs at most $O(n^3)$ time (refer to [5]). Line 2 needs at most $O(n^2)$ time as it scans at most once for each node to find its matching node, and determine if the matched nodes can be combined. Line 3 needs $O(n)$ time as it scans the nodes once for each partition to form the clusters. Lines 5~6 has the complexity of $O(n^2)$ because, for each matched node, it needs no more than once scanning to calculate the formula and to combine to its matched one to form a node of the next level coarser graph. So the total complexity of lines 4~6 is limited by $O(n^3)$. Both line 7 and line 8 need $O(n)$ time to complete. Line 9 needs $O(n^2)$ because, for each node, it needs at most one scan to all nodes to update edges, while calculating weights

allow this type of match in order to combine two isolated partition of smaller sizes provided their combination does not violate the rule of formula (1).

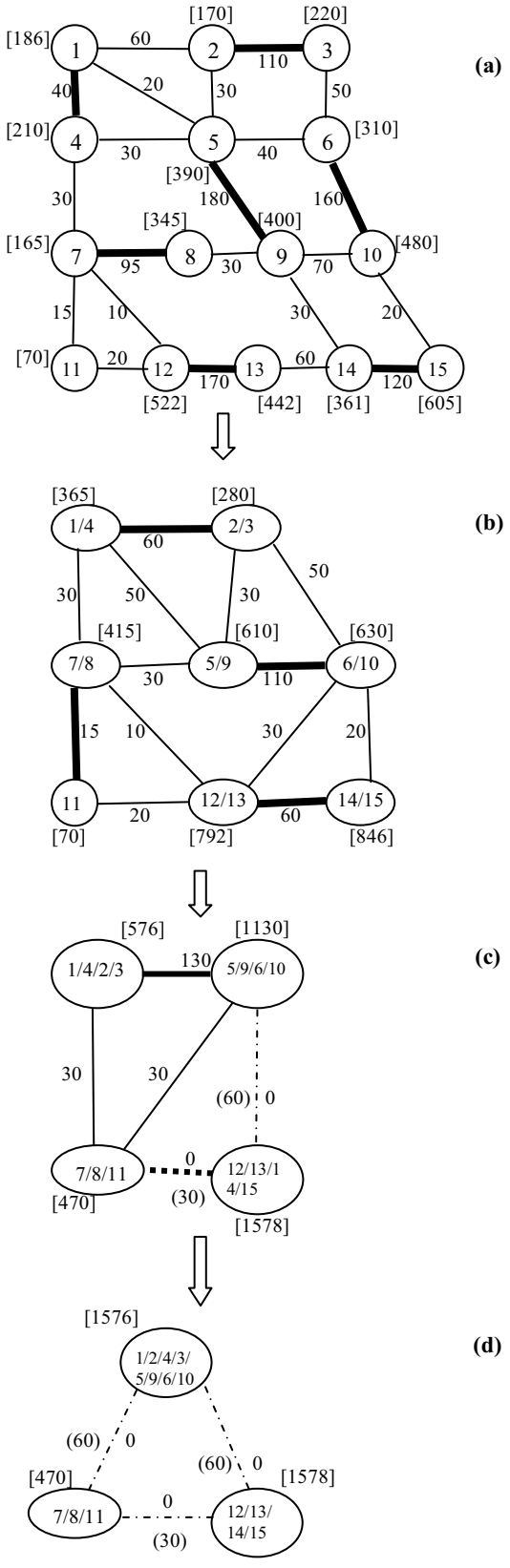

Figure 2. Examples of matching and combination process

of edges between the newly formed nodes of the next level of coarsened graph. Line 10 completes the recursive execution of the algorithm.

Let *g(n)* be the complexity function of the algorithm. We analyse both the best case and the worst case. First, consider the best case of the execution (i.e., all nodes in *G* were matched in step [1]). Denote *f(n)* as the complexity of this case. For an ideal matching, each node is matched, thus the next level of coarser graph has *n*/2 nodes. In this case, we can get a recurrence formula for the complexity function as

$$f(n) = \begin{cases} 0 & \text{if } n \le 1 \\ \alpha \cdot n^3 & \text{if } n>1 \text{ and no further match exists} \\ \alpha \cdot n^3 + f(n/2) & \text{if } n>1 \text{ and further match exists} \end{cases}$$

where α is a constant. As the collapsing reduces half the number of available (multi)nodes, either $n \le 1$ or ($n>1$ *and no further match exists*) will become true after running the algorithm recursively for some rounds. Therefore, for a large *n*, according to the recurrent property of *f(n)*, we have

$$f(n) = \alpha \cdot n^3 + f(n/2) = \alpha \cdot n^3 + \alpha \cdot (n/2)^3 + f(n/4) = \ldots$$
$$= \alpha \cdot (n^3 + (n/2)^3 + (n/4)^3 + \ldots + 1) + f(1)$$

This equation is valid for any *n* that is a power of 2, say $n = 2k$. Thus, we have

$$f(n) = \alpha n^3 \cdot (1 + 1/2^3 + 1/2^{2\cdot 3} + 1/2^{3\cdot 3} + \ldots + 1/2^{3(k-1)}) + f(1)$$

Recalling that $f(1) = 0$, we get $f(n) = 8/7 \cdot \alpha \cdot n^3$, or

$$f(n) = O(n^3) \qquad (6)$$

If *n* is not a power of 2, there must exist *k* such that $2k < n \le 2(k+1)$. Therefore, we have $8/7 \cdot \alpha n^3 \le f(n) \le 8/7 \cdot 8\alpha n^3$, which still leads to formula (6).

Secondly, consider the case where some unmatched nodes produced in step [1], or some matched nodes are not be able to combined in lines 4~7. Denote *F(n)* as the complexity of the algorithm in this case. Without loss of generality, we assume that the average number of matching pairs is $n/4$ ($\approx (1 + 2 + \ldots + n/2)/(n/2)$). After collapsing, the next level of coarser graph will have $3n/4$ multinodes. In this case, we can get a recurrent function as

$$F(n) = \begin{cases} 0 & \text{if } n \le 1 \\ \alpha \cdot n^3 & \text{if } n>1 \text{ and no further match exists} \\ \alpha \cdot n^3 + F(3n/4) & \text{if } n>1 \text{ and further match exists} \end{cases}$$

where *F(3n/4)* is the complexity of the matching and collapsing process on the next level coarser graph. In the worst case, every recursive execution of the algorithm would produce a (next level) coarser graph whose number of multinodes is about four thirds of that of the current graph. After *k* times of recursive execution, either the number of the (multi)nodes of the graph becomes 1, or no further match can be found from the graph. So, for simplicity, we can assume that

$n \cdot (\frac{3}{4})^k = 1$, or $n = \left\lfloor (\frac{4}{3})^k \right\rfloor$ for some k. According to the recurrent relation of $F(n)$, we can derive

$$F(n) = \alpha \cdot n^3 + F(\frac{3}{4}n) = \alpha \cdot n^3 + \alpha \cdot (\frac{3}{4}n)^3 + F(\frac{3^2}{4^2}n)$$

$$= \ldots \le \alpha \cdot n^3 (1 + \frac{3^3}{4^3} + \frac{3^6}{4^6} + \frac{3^9}{4^9} + \ldots) = \frac{64}{37} \cdot \alpha \cdot n^3$$

or

$$F(n) \le O(n^3) \quad (7)$$

Following the discussion above, it is not difficult to prove that formula (7) holds for any n (the proof is omitted here).

As the complexity of our algorithm is always greater than or equal to $f(n)$, and less than or equal to $F(n)$, i.e., $O(n^3) = f(n) \le g(n) \le F(n) \le O(n^3)$, we get

$$g(n) = O(n^3).$$

5. Simulations

The simulation work is to demonstrate the reduction of the I/O costs in spatial join processing by using the matched based data partitioning method against that proposed in [1]. The simulations are conducted with artificial and real world spatial data sets.

Table 1: A comparison: numbers of clusters produced by the matrix-based and match-based partitioning methods

	Average No. of clusters	
Size of candidate set	Matrix-based	Match-based
200	2	2
400	3	3
800	9	8
1200	16	14
1600	37	33
2000	59	49
2400	86	68

In the simulation, two criteria are used to measure the quality of the clusters produced. For the same candidate sets, we first compare the numbers of clusters produced using the two methods, and then compare total size of overlapping objects between clusters produced by the two methods, respectively. In the simulations, the memory capacity (i.e., q as in formula (1)) varies from 500 to 2000 blocks, and the sizes of the spatial objects vary from 8 to 1654 vertices, with average size of 92 vertices. The following tables show that when the size the candidate set is not vary large, (i.e., hundreds to a thousand), the two methods produced similar results. However, when the candidate set becomes large, the matched based method outperforms the matrix-based method significantly. The total size of overlapping objects between clusters produced by the two methods is almost linear to the number of clusters produced, and thus was not given here. Table 1 shows the simulation results.

6. Conclusion

The spatial join is one of the most important operations in spatial databases. The cost of spatial join could be very high due to the large sizes of spatial objects and the computation-intensive spatial operations. In spatial join processing, spatial objects are usually partitioned into clusters and then processed cluster by cluster. The partitioning step aims to partition objects into clusters such that objects inside a cluster are more likely to join with each other than those across clusters.

This paper proposed a match-based spatial objects partitioning method. We demonstrated that the new methods outperform the existing methods. Simulation results have shown that the match based partitioning method can produce less number of clusters than the method proposed in [1].

References

[1] Jitian Xiao, Yanchun Zhang & Xiaohua Jia. Clustering Non-uniform-sized Spatial Objects to Reduce I/O Cost for Spatial Join Processing, *The Computer Journal*, Vol. 44, No.5, 2001.

[2] H. Samet and Walid Aref, Spatial Data Models and Query Processing. *Modern Database Systems*, Addison-Wesley Publishing Company, Inc, 1995.

[3] Y. Theodoridis, E. Stefanakis, T. Sellis. Cost Model for Join Queries in Spatial Databases. *Proc. of ICDE'98*, Orlando, Florida, USA, 1998.

[4] T. Brinkho, H. Kriegel and Bernhard Seeger, Efficient Processing of Spatial Joins Using R-trees. *Proc. ACM SIGMOD Int. Conf. on Management of Data,* pp.237-246, 1993.

[5] E. L. Lawler, *Combinatorial Optimization: Net-works and Matroids*. Holt, Rinehart and Winston, New York, 1976.