2006

A Comparison of Heuristics for Scheduling Spatial Clusters to Reduce I/O Cost in Spatial Join Processing

Jitian Xiao
Edith Cowan University

10.1109/ICMLC.2006.258779

This is an Author's Accepted Manuscript of: Xiao, J. (2006). A Comparison of Heuristics for Scheduling Spatial Clusters to Reduce I/O Cost in Spatial Join Processing. Proceedings of International Conference on Machine Learning and Cybernetics. (pp. 2455-2460). Dalian, China. IEEE. Available here

A COMPARISON OF HEURISTICS FOR SCHEDULING SPATIAL CLUSTERS TO REDUCE I/O COST IN SPATIAL JOIN PROCESSING

JI-TIAN 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:

In spatial join processing, a common method to minimize the I/O cost is to partition the spatial objects into clusters, and then to schedule the processing of the clusters in the spatial join processing such that the number of times the same objects to be fetched into memory can be minimized. A key issue of this clustering-and-scheduling approach is how to produce a better sequence of clusters to guide the cluster scheduling thus to reduce the total I/O cost of spatial join processing. This paper describes three cluster sequencing heuristics. An extensive comparison among them has been conducted, and simulation results have shown that, while using the cluster sequences generated to guide the cluster scheduling can significant reduce the I/O cost in fetching spatial objects in spatial join processing, their performance differs.

Keywords:

Spatial databases; Spatial join processing; maximum spanning tree; Ant colony optimization; scheduling, Match

1. Introduction

In Spatial databases, the cost of spatial join processing could be very high due to the large sizes of spatial objects and the computation-intensive spatial operations [4]. 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 [1]). The first step chooses pairs of spatial objects that are likely to satisfy the join predicate. The second step examines the predicate satisfaction for all those pairs of objects 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.

It is very important to reduce the I/O cost at the refinement step. Experiments have shown that I/O cost (i.e., disk accesses) at the refinement step takes significant amount of time, as compared with the CPU time for spatial join [3]. If a large number of spatial objects are involved in a spatial join operation, a common method to minimize the I/O cost is to partition the spatial objects into clusters, and then to schedule the processing of the clusters in the spatial join processing such that the number of times the same objects to be fetched into memory can be reduced. The key issue of cluster scheduling is how to produce better scheduling sequences of the clusters such that the total I/O cost is minimized. However, the problem of finding a best spatial cluster scheduling sequence is NP-complete [7]. In our previous work, we have developed a number of heuristics [5, 6, 7] to produce spatial cluster scheduling sequences. This paper is to evaluate some individual heuristics and conduct comprehensive comparison among them.

The rest of the paper is organized as follows: In Section 2, we formalize the spatial cluster scheduling problem. In Section 3, some preliminary concepts are presented and some cluster sequencing heuristics are reviewed. An example is given in Section 4 to show the performance of individual heuristics. Simulation results are presented in Section 5. And Section 6 concludes the paper.

2. Formulation of cluster scheduling

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

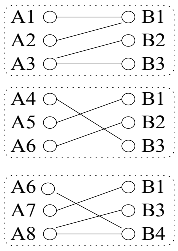
1-4244-0060-0/06/$20.00 ©2006 IEEE

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.

S_id	T_id
A1	B1
A2	B1
A3	B2
A3	B3
A4	B3
A5	B1
A6	B2
A6	B4
A7	B1
A8	B3
A8	B4



(a) A candidate set. (b) A candidate clustering.

Figure 1. An example of a candidate set and its clustering

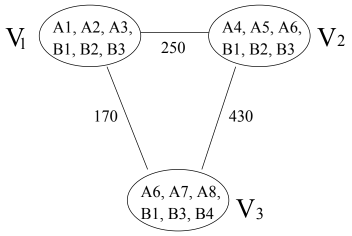
The *refinement* step is to perform $S \chi T$ on the pairs of objects indexed by F to produce the final join results. At this step, the S and T objects need to be fetched into the main memory for the full spatial join check. Since the memory size is limited and it can not keep all objects of F in memory at the same time, the objects need to be partitioned into clusters. Objects in the same clusters are brought into the memory together and processed in a batch. For example, Figure 1 (b) is a partitioning of the candidate set shown in Figure 1 (a).

Assume that the spatial objects referenced in F have been partitioned into clusters. Our goal is to schedule the clusters in a way such that the repeatedly fetch of the overlapping objects between consecutive clusters is minimized. The I/O cost, in this paper, is measured in terms of the size of object data (e.g., number of vertices of the spatial object) that are fetched into the memory for the refinement operation.

Let $\mathbb{V} = \{v_1, v_2, ..., v_k\}$ be the set of objects referenced in F, and $V_1, V_2, \ldots, V_n$ the clusters of $\mathbb{V}$。 For each i ($1\le i\le n$), $V_i=\{v_{i_1}, v_{i_2}, ..., v_{i_m}\}$ ($m\ge 1$), $v_{i_j} \in \mathbb{V}$ ($1\le j\le m$). That is, $\bigcup_{i=1}^{n} V_i = \mathbb{V}$ and $V_i \neq \phi$ for each i ($1\le i\le n$). For convenience, we define $size(V_i)$ as the sum of the sizes of objects in V_i, i.e., $size(V_i)=\sum_{v\in V_i} s(v)$ where $s(v)$ is the size of object v.

We introduce a weighted graph $G = (V, E, w)$, upon $\mathbb{V}$, called *cluster overlapping* (*CO*) graph, to represent the overlapping relationships between data clusters. The node set $V = \{V_1, V_2, ..., V_n\}$ is a set of clusters, and the edge set E is defined as: for each pair of nodes V_i and V_j ($i \neq j$), there is an edge $E_{ij} = (V_i, V_j)$ if $w(V_i, V_j) = size(V_i \cap V_j) \neq 0$. Here $w(V_i, V_j)$, also denoted as $w(E_{ij})$, is the weight of edge $E_{ij.}$ For instance, based on the object sizes given in Figure 2 (a), Figure 2 (b) shows the CO graph corresponding to the clusters in Figure 1 (b).

oid	obj size
A1	200
A2	80
A3	40
A4	30
A5	32
A6	260
A7	18
A8	60
B1	60
B2	80
B3	110
B4	76



(a) Object size (b) cluster overlapping graph

Figure 2. An example of CO graph

At refinement step, if the object clusters are processed in the sequence of $V_1, V_2, ..., V_n$ (i.e., no scheduling), then the total I/O cost is:

$$C_{I/O}=\sum_{i=1}^{n} size(V_i)-\sum_{i=1}^{n-1} size(V_i \cap V_j) \tag{1}$$

When processing cluster V_{i+1}, objects in $V_i \cap V_{i+1}$ are already in memory just after processing V_i. There is no need to load these objects again.

Generally, for a schedule π which determines the processing sequence of $V_1, V_2, ..., V_n$ as $V_{\pi_1}, V_{\pi_2}, ..., V_{\pi_n}$, where $V_{\pi_i} \in V$ and $V_{\pi_i} \neq V_{\pi_j}$ for $i \neq j$, the I/O cost for schedule π is

$$\begin{aligned} C^{\pi}_{I/O} &= size(V_{\pi_1})+\sum_{i=1}^{n-1}(size(V_{\pi_i})-size(V_{\pi_i}\cap V_{\pi_{i+1}})) \\ &= \sum_{i=1}^{n} size(V_{\pi_i})-\sum_{i=1}^{n-1} size(V_{\pi_i}\cap V_{\pi_{i+1}}). \end{aligned} \tag{2}$$

When the clusters are given, $\sum_{i=1}^{n} size(V_{\pi_i})$ is a constant. Let y be:

$$y=\sum_{i=1}^{n-1} size(V_{\pi_i}\cap V_{\pi_{i+1}}) \tag{3}$$

The objective of spatial cluster scheduling is to find a schedule π such that y is maximized, which is the case that $C^{\pi}_{I/O}$ is minimized.

3. Spatial Cluster Scheduling Heuristics

Given a CO graph $G = (V, E, w)$ with $V = \{V_1, V_2, ..., V_n\}$, an *maximum overlapping (MO) order* among $V_1, V_2, ..., V_n$ is a sequence ($V_{i_1}, V_{i_2}, ..., V_{i_n}$) such that $\sum_{l}^{n-1} size(V_{i_l} \cap V_{i_{l+1}})$ reaches the maximum among all permutations of V [7]. In other words, an MO order in a CO graph G is a permutation of nodes in G such that the total size of overlapping objects between adjacent nodes reaches the maximum. For example, (V_1, V_2, V_3) is an MO order in CO graph in Figure 2(a), and the total size of overlapping objects between adjacent nodes in the order is 680.

The simplest algorithm to find an MO order is to check all permutations of V to see which one makes the max $\{ \sum_{l}^{n-1} size(V_{i_l} \cap V_{i_{l+1}}) \}$. The complexity of the method clearly has factorial order and is certainly not practical.

Although an MO order exists for each CO graph G, it is impossible to find an MO order in polynomial time. However, the task of finding an MO order can be reduced to the case where G is a connected graph [5]. We now describe three heuristics that produce *approximation of MO* (AMO) order in given CO graph G.

3.1. Maximum Spanning Tree Based Heuristic

A maximum spanning tree (MST) based heuristic was developed in [7] to produce an AMO order of relative "high" overlapping weights in the sense that the weight of the AMO order produced by the algorithm is always greater than or equal to half the weight of an optimal MO order. The algorithm consists of three steps: The first step produces a maximum spanning tree T of the CO graph G; the second step conducts a *depth-first search* (DFS) on T and, in the third step, an AMO order is built, which is the traversal order of the DFS on T. The complexity of the algorithm is $O(m^2 \log_2 m)$, where $m = max(|V|, |E|)$.

3.2. ACO-based Heuristic

The *ant-colony optimization* (ACO) based meta-heuristic is a population-based approach to the solution of discrete optimization problems. It imitates real ants searching for food, i.e., by finding the shortest path from a food source to their nest. Ants use an aromatic essence, called *pheromone*, to communicate information regarding the food source. While ants move along, they lay pheromone on the ground which stimulates other ants rather to follow that trail than to use a new path. As other ants observe the pheromone trail and are attracted to follow it, the pheromone on the path will intensified and reinforced and will therefore attract even more ants.

The typical application of ACO is the travelling salesman problem (TSP) [2], defined as follows: A graph G=(V, E, w) with node set V and edge set E is given; each edge $e \in E$ has a weight w(e) associated, representing the length of it. The problem is to find a minimal-length closed tour that visits all the nodes once and only once. In the ACO approach each edge of the graph has two associated measures: the heuristic desirability η_{ij}, which is defined as the inverse of the edge length and never changes for a given problem instance, and the pheromone trail τ_{ij}, which is modified at runtime by ants. Each ant has a starting node and its goal is to build a solution, that is, a complete tour. A tour is built node by node: when ant k is in node i it chooses to move to node j using a probabilistic rule that favors nodes that are close and connected by edges with a high pheromone trail value. Nodes are always chosen among those not yet visited in order to enforce the construction of feasible solutions. Then pheromone trail is updated on the edges of the solutions.

To apply the ACO approach for finding an AMO order for an arbitrary CO graph $G=(V, E, w)$, we extended G to a complete graph $G'=(V, E', w')$ by the following steps. For any pair of nodes $v_i, v_j \in V, 1 \leq i, j \leq n$, add an edge (v_i, v_j) to E'. If $(v_i, v_j) \in E$, then define $w'(v_i, v_j) = 1/w(v_i, v_j)$; otherwise define $w'(v_i, v_j) = w_{max}$, where w_{max} is a very large number. $w'(v_i, v_j)$ is taken as the length between nodes v_i and v_j.

Then the ACO algorithm is applied to G' to find a TSP solution, which is a shortest close tour on G'. Once a TSP solution was found, an AMO order can be determined by simply removing an edge of maximum weight from the solution and taking the order of the resultant path as the AMO order [5]. The complexity of the ACO-based algorithm is $O(Nc \cdot n^3)$, where Nc is the predetermined iteration parameter of the ACO algorithm (Nc usually takes a value of n).

3.3. Match-based Heuristic

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 (edge weighted) 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 [2] and the complexity of his algorithm is $O(n^3)$, where n is the number of nodes of G. For any graph, Edmonds's algorithm outputs a *maximal match*[1] of G.

[1] 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

The basic idea behind the match-based algorithm [6] is first to divide the CO graph into sets of disjoint *path graphs*[2] such that the sum of the edge weights of the longest paths in the path graphs reaches the maximum, and then link these paths using maximal match among the endpoints of the longest paths in the path graphs.

The match-based heuristic is a recursive one containing three main steps: In the first step, a maximal match M of G is produced using *Edmonds*'s algorithm (for details see [2]). Edges in M are taken as the initial AMO order. Then G is divided into sets of path graphs, each consists of a pair of matched nodes and an edge connecting the matched nodes. Each unmatched node of G forms a special path graph, i.e., one without an edge. In the second step, the graph G is coarsened by collapsing the matching nodes. At this step, each pair of matching nodes are combined to form a single node of the next level coarser graph $G' = (V', E', w')$. Nodes in V' are either in the form of $v = \{v_i, v_j\}$, where $v_i, v_j \in V$ are matched in M, i.e., $(v_i, v_j) \in M$, or $v = \{v_i\}$, where v_i is a unmatched node of M. That is, $V' = \{\{v_i, v_j\} \mid v_i, v_j \in V$ and $(v_i, v_j) \in M\} \cup \{\{v_i\} \mid v_i$ is a unmatched node of $M\}$.We refer to node v of form $\{v_i, v_j\}$ in V' a *multinode*. E' and w' are then defined such that the edge between any pair of multinodes v' and v'' corresponds to an edge in E whose two endpoints are in v' and v'', respectively, and whose weight is maximal among all edges connecting nodes in between the multinodes v' and v'', if such an edge exists (note that any pair of unmatched nodes is not connected in both G and G'). After graph G' is built, *Edmonds*'s algorithm is applied to G' again to produce a maximal match M'.

The above matching and collapsing process continues until no further matching can be found. Then in the third step, the heuristic produces the AMO order according to the output of the above procedure.

Intuitively, if we conceptually take a pair of matching nodes and the edge between them as a path graph at the end of first round of matching and collapsing process, then from the second round of matching and collapsing process on, these path graphs are merged pairwisely in a way that their longest paths are linked end by end using an edge of maximal weight between endpoints of the paths. With the matching and collapsing process going on, paths are linked using the maximal matching on levels of coarser graphs until a set of disconnected path graphs is reached. At this stage, a sequence of nodes of the longest path for each path graph was output. Any order of these sequences can be taken as an AMO, because the produced path graphs are non-joint with each other, and each node of the original CO graph belongs one and only one path graph.

4. Comparison among heuristics: An Example

Now let us compare the performance of these heuristics using the example given in [4], which is re-produced as below: Let $\mathbb{V} = \{a_1, a_2, ..., a_{36}\}$ be a spatial object set and $V = (V_1, V_2, ..., V_6)$ a set of clusters on $\mathbb{V}$. The relationship between an object a_i ($1 \le i \le 36$) and a cluster V_j ($1 \le j \le 6$) is given by the following incidence matrix (m_{ij}), i.e., $m(i, j)=1$ if $a_j \in V_i$, and $m(i, j)=0$ otherwise.

V_i	V_1	V_2	V_3	V_4	V_5	V_6	V_i	V_1	V_2	V_3	V_4	V_5	V_6
a_1	1	1	1	1	0	0	a_{19}	1	0	0	0	1	0
a_2	1	1	1	1	0	0	a_{20}	1	0	0	0	1	0
a_3	0	1	0	1	0	0	a_{21}	1	0	0	0	1	0
a_4	0	1	0	1	0	0	a_{22}	0	0	1	0	1	1
a_5	0	1	0	1	0	0	a_{23}	0	0	1	0	1	1
a_6	0	1	0	1	0	0	a_{24}	0	0	1	0	1	1
a_7	0	1	0	1	0	0	a_{25}	0	0	0	0	1	1
a_8	0	1	0	1	0	0	a_{26}	0	0	0	0	1	1
a_9	0	1	1	0	0	0	a_{27}	0	0	0	0	1	1
a_{10}	0	1	1	0	0	0	a_{28}	0	0	0	0	1	1
a_{11}	0	1	1	0	0	0	a_{29}	0	0	0	0	1	1
a_{12}	0	1	1	0	0	0	a_{30}	0	0	0	0	1	1
a_{13}	0	0	1	0	0	0	a_{31}	0	0	0	1	0	1
a_{14}	0	0	1	0	0	0	a_{32}	0	0	0	1	0	1
a_{15}	1	0	1	1	0	0	a_{33}	0	0	0	1	0	1
a_{16}	1	0	1	1	0	0	a_{34}	0	0	0	1	0	1
a_{17}	1	0	0	1	0	0	a_{35}	0	0	0	1	0	1
a_{18}	1	0	0	0	1	0	a_{36}	0	0	0	1	0	1

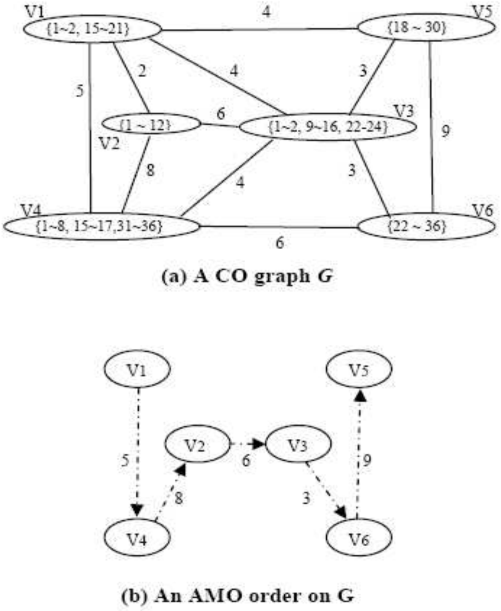
The corresponding CO graph is given in Figure 3 (a). In this example, the sizes of all objects are identical, thus are not important. For simplicity, an object a_i is expressed by its index i in the figure, and the size of a_i is taken as 1, for all $1 \le i \le 36$. By applying the MST based algorithm to the CO graph, an AMO order was produced as shown in Figure 3 (b), which is V_1, V_4, V_2, V_3, V_6, V_5, with the total overlapping weight 31.

By applying the match-based algorithm to the CO graph in Figure 3(a), the first step produces a maximal match $M = \{(V_1, V_3), (V_2, V_4), (V_5, V_6)\}$. That is, the initial set of path graphs contains three path graphs, with path sequences $P_1 = (V_1, V_3)$, $P_2 = (V_2, V_4)$ and $P_3 = (V_5, V_6)$, respectively. After collapsing, the next round of matching produces a maximal match containing one edge, i.e., $(\{V_1, V_3\}, \{V_2, V_4\})$, and an isolated node which is the multinode $\{V_5, V_6\}$. By collapsing the (matched) multinodes, two path graphs (i.e., those with path sequences P_1 and P_2) were

sum of the edge weights of the match is maximal among all matches of the graph.

[2] A *path graph* $G = (V, E)$ with n nodes is a graph in which all nodes in V can be listed as a sequence $v_1, v_2, ..., v_n$ such that $(v_1, v_2), (v_2, v_3), ... (v_{n-1}, v_n)$ are the only edges of E.

merged to form a new path graph, with longer path sequence V_1, V_3, V_2, V_4. The endpoints (i.e., V_1 and V_4) of this path sequence form a new multinode in the next level coarser graph. The next matching and collapsing process results in a single multinode $\{V_1, V_5\}$ that makes the algorithm stop. This step merges the two path sequence (V_1, V_3, V_2, V_4) and (V_5, V_6) together by inserting the edge between the matching nodes V_4 and V_6, leaving the final endpoints of the longest path to be V_1 and V_5. The final AMO order is V_1, V_3, V_2, V_4, V_6, V_5, as shown in Figure 4(a).



(a) A CO graph *G*

(b) An AMO order on G

Figure 3. A CO graph and its AMO order produced by the MST based algorithm.

By comparing the above AMO orders, we found that the match based algorithm produced a better AMO order by the fact that the total overlapping weight of the AMO order produced by the match based algorithm is 33, which is the optimal MO order in this example, while it is 31 for the AMO order produced by the MST algorithm.

When applying the ACO-based heuristic to the CO graph in Figure 3(a), two AMO orders were produced, as shown in Figure 4.   one is the same as that in Figure 4(a), and the other is as in Figure 4(b). Both have a total overlapping weight of 33, the same with that produced by the match-based algorithm.

5. Simulations

The simulation work is to demonstrate the reduction of the I/O costs in spatial join processing by using different AMO orders to guide the scheduling of processing of clustered join operations. Two types of simulations were conducted. The first type of simulations is to show the I/O cost reduced by using various cluster sequencing methods, and the second is to compare the overlapping weights on AMOs produced by various heuristics.

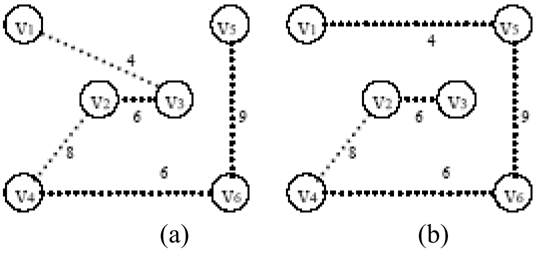


Figure 4. AMO order produced by (a) match-based and (b) ACO-based heuristics.

In the first type of simulations, AMO orders produced by MST-, match- and ACO-based heuristics are applied to the same application scenarios to guide spatial cluster scheduling. For ease of description, the related schedules are called MST, MB, and ACO, respectively, in this section. These schedules are simulated against each other using the following spatial cluster scheduling strategy: the schedule fetches spatial objects into the memory, cluster by cluster, in the AMO orders produced by the MST-based, match-based and ACO-based heuristics, respectively. The overlapping objects between two consecutive clusters are not fetched into the memory again when processing the next cluster. Although match-based and ACO-based heuristics take longer time than MST-base heuristic does in finding AMO orders, their calculation costs can all be neglected, when comparing with the total fetching cost.

In the simulations, most spatial datasets are generated while a small portion of datasets is from real spatial applications. The object sizes change from tens to hundreds of vertices. At each simulation point, the simulation runs 10 times. Since every object needs to be fetched into the memory for the refinement operation, for simplicity, we measure the I/O cost in terms of the total size of the overlapping objects that are fetched repeatedly into the memory for processing (i.e., y value in formula (3)). The I/O costs of Y-axis are the mean values of y in all runs.

Figure 5 shows the I/O cost versus the cluster numbers. The average size of clusters in this figure is 20. We can see that ACO and MB are quite similar in performance, and they perform all the time better than MST. On average, ACO and MB can achieve over 16% saving when comparing with MST. For example, when the number of clusters is 50, the average size of overlapping objects to be fetched repeatedly into memory by MST is 2069, while it is 1760 by ACO and 1766 by MB, respectively. As the number of clusters grows, the performance gain obtained by ACO and MB, respectively, gets greater. When cluster size changes, similar trends are achieved in our simulations. Due to space limitation, the results are omitted here.

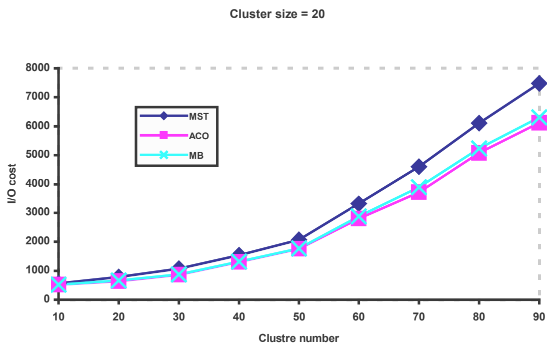


Figure 5. I/O cost versus cluster numbers (cluster size=20).

Table 1. A comparison of overlapping weights among AMO orders produced by three heuristics

size (n,e) / *heuristic*	20/50	30/60	40/100	50/120	100/200
ACO-based	2399	11898	27276	27182	47673
Match-based	2375	11825	27577	26287	47576
MST-based	2191	11161	25576	25425	46439

Table 1 shows the average overlapping weights among AMO orders produced by ACO-based, match-based and MST-based heuristics, respectively, for graphs of size (n, e), where n is the number of nodes, and e the number of edges. We observe that, for any size of graphs, the average weight of the match-based AMO orders is always greater than (or equal to) that of the MST-based AMO orders, and it is very close to that of ACO-based AMO orders. As ACO produces optimal or near-optimal solution for most NP-hard problems, the average weight of the AMO order produced by the ACO-based algorithm is very close to that of the MO order. Therefore, the AMO orders produced by match-based algorithm are very close to the MO order.

However, from the simulation, the computation cost of ACO-based AMO order is sensitive to the number of edges of the CO graph, while MB and MST are not. Due to this, ACO is not suitable for online spatial join service where spatial join processing must be completed in a reasonable time limit. On the other hand, for offline spatial join, ACO performs better than the other two schedules.

6. Conclusion

In spatial join processing, spatial objects are usually partitioned into clusters and then are processed cluster by cluster. Since two clusters may have overlapping, the overlapping objects may be repeatedly loaded into memory. It is important to schedule the processing of the clusters in such a sequence that two consecutive clusters in the sequence have higher number of overlapping objects, thus, there is no need to load those overlapping objects when processing the next cluster because they are already in the memory. The I/O cost can, therefore, be reduced.

The key issue behind the spatial cluster scheduling method is how to produce a better AMO order to guide the scheduling. This paper described three cluster-sequencing heuristics. A comparison among them has been conducted to evaluate their performance. Simulation results have shown that, while ACO-based and match-based heuristic produce better AMO orders than the MST-based one does, ACO is not suitable for online spatial join processing.

References

[1] L. Becker, A. Giesen, K. Hinrichs and J. Vahrenhold. Algorithms for Performing Polygonal Map Overlay and Spatial Join on Massive Data Sets. R. G¨uting, D. Papadias, F. Lochovsky (Edt.): SSD'99, LNCS 1651. pp.270-285. Springer-Verlag Berlin Heidilberg, 1999.

[2] E. L. Lawler, Combinatorial Optimization: Networks and Matroids. Holt, Rinehart and Winston, New York, 1976.

[3] M. L. Lo and C. V. Ravishankar. Spatial Joins Using Seeded Tree, Proc. ACM SIGMOD Int. Conf. on Management of Data, pp209-220, 1994

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

[5] J. Xiao, Applying the Ant Colony Optimization Algorithm to the Spatial Cluster Scheduling Problem. Proceedings of the 3rd International conference on Machine Learning and Cybernetics (ICMLC04), Shanghai, China, August 26-29, 2004, pp1341-1346.

[6] Jitian Xiao, Match Based SJP Cluster Sequencing and Scheduling in Spatial Databases, Proceedings of the 2nd Computational Intelligence, Robotics and Autonomous Systems, Singapore, Dec.15-18, 2003.

[7] J. Xiao, Y. Zhang, X. Jia and X. Zhou. A Schedule of Join Operations to Reduce I/O Cost in Spatial Database Systems, Data & Knowledge Engineering, Elsevier Science B.V, Vol. 35, 2000, pp299-317.