

2006

Use of a weighted matching algorithm to sequence clusters in spatial join processing

Husen Husen
Edith Cowan University

Follow this and additional works at: https://ro.ecu.edu.au/theses_hons



Part of the [Numerical Analysis and Scientific Computing Commons](#)

Recommended Citation

Husen, H. (2006). *Use of a weighted matching algorithm to sequence clusters in spatial join processing*.
https://ro.ecu.edu.au/theses_hons/1413

This Thesis is posted at Research Online.
https://ro.ecu.edu.au/theses_hons/1413

Edith Cowan University

Copyright Warning

You may print or download ONE copy of this document for the purpose of your own research or study.

The University does not authorize you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site.

You are reminded of the following:

- Copyright owners are entitled to take legal action against persons who infringe their copyright.
- A reproduction of material that is protected by copyright may be a copyright infringement. Where the reproduction of such material is done without attribution of authorship, with false attribution of authorship or the authorship is treated in a derogatory manner, this may be a breach of the author's moral rights contained in Part IX of the Copyright Act 1968 (Cth).
- Courts have the power to impose a wide range of civil and criminal sanctions for infringement of copyright, infringement of moral rights and other offences under the Copyright Act 1968 (Cth). Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.

USE OF THESIS

The Use of Thesis statement is not included in this version of the thesis.

Use of a Weighted Matching Algorithm to Sequence Clusters in Spatial Join Processing

**EDITH COWAN UNIVERSITY
LIBRARY**

A thesis submitted in partial fulfilment of the requirements for the
degree of

Bachelor of Science (Software Engineering) Honours

By: Husen Husen
Student ID: 2041002

Faculty: Computing, Health and Science
Institution: Edith Cowan University

Supervisors: Dr. Jitian Xiao and Michael Collins

Date of submission: December 2006

Abstract

One of the most expensive operations in a spatial database is spatial join processing. This study focuses on how to improve the performance of such processing. The main objective is to reduce the Input/Output (I/O) cost of the spatial join process by using a technique called cluster-scheduling. Generally, the spatial join is processed in two steps, namely filtering and refinement. The cluster-scheduling technique is performed after the filtering step and before the refinement step and is part of the housekeeping phase. The key point of this technique is to realise order wherein two consecutive clusters in the sequence have maximal overlapping objects.

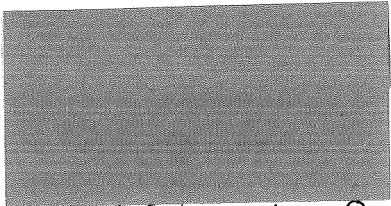
However, finding the maximal overlapping order has been shown to be Non-deterministic Polynomial-time (NP)-complete. This study proposes an algorithm to provide approximate maximal overlapping (AMO) order in a Cluster Overlapping (CO) graph. The study proposes the use of an efficient maximum weighted matching algorithm to solve the problem of finding AMO order. As a result, the I/O cost in spatial join processing can be minimised.

Declaration

I certify that this thesis does not, to the best of my knowledge and belief:

- (i) incorporate without acknowledgment any material previously submitted for a degree or diploma in any institution of higher education.
- (ii) contain any material previously published or written by another person except where due reference is made in the text; or
- (iii) contain any defamatory material.

I also grant permission for the Library at Edith Cowan University to make duplicate copies of my thesis as required.

Signature: 

Date: 1 22 March 07

Acknowledgements

I am most grateful to my two supervisors: Dr. Jitian Xiao and Michael Collins. Michael assisted me with the application of honours study though it was very late to apply for honours study at that time. He also provided invaluable feedback for my honours proposal and thesis. Jitian was always very helpful for the duration of my research. He provided support and guidance for my research. He was the most important resource in my study.

Thanks also go to my lecturer Judy Clayden. Judy was very kind and willing to help me with the grammar checking of my research proposal.

The reviewers of my research proposal have provided valuable feedback. Thanks to them as well. Also, a big thank you to my fellow honours students for their sharing of experience.

Most importantly, I thank my family: Mum, Dad, sisters, and brothers for their support and patience. I am very lucky to have you all during this very tough time, not to mention the other times.

Table of Contents

| | | |
|-------|--|----|
| 1 | Introduction | 1 |
| 1.1 | The Background to the Study..... | 1 |
| 1.1.1 | Spatial Join Processing..... | 4 |
| 1.2 | The Significance of the Study | 5 |
| 1.3 | The Purpose of the Study | 6 |
| 1.4 | Statement of Research Questions..... | 7 |
| 2 | Review of Relevant Literature | 9 |
| 2.1 | General Literature Review | 9 |
| 2.1.1 | Filter Step | 9 |
| 2.1.2 | Refinement Step | 14 |
| 2.2 | Literature on Previous Findings..... | 17 |
| 2.2.1 | Sequencing/Scheduling Strategies | 18 |
| 2.2.2 | Clustering Strategies | 24 |
| 2.3 | Specific Studies Similar to the Current Study..... | 27 |
| 3 | Theoretical Framework | 30 |
| 3.1 | Identification of Variables Impacting On the Research Questions..... | 30 |
| 3.2 | Identification of Assumptions Underpinning the Study | 30 |
| 4 | The Proposed Method | 33 |
| 4.1 | Maximal Weight Matching (MWM) Algorithm..... | 33 |
| 4.2 | The Complexity of MWM Algorithm..... | 36 |
| 4.3 | Comparison with Other Methods | 38 |
| 5 | Material and Methods | 45 |
| 5.1 | Design and Procedure of the Study..... | 45 |
| 5.2 | Description of Instruments Employed | 46 |
| 5.3 | Data Analysis | 46 |
| 6 | Results and Findings | 48 |
| 7 | Further Study..... | 51 |
| 8 | Conclusion | 52 |
| | References | 53 |
| | Appendix A – Definitions of Terms | 56 |
| | Appendix B – $O(n^3)$ Weighted Matching Algorithm..... | 58 |
| | Appendix C – Experimental Results | 61 |
| | Appendix D – CD Contents | 65 |

List of Figures

| | |
|---|----|
| Figure 1: Constructing a join index from two relations | 2 |
| Figure 2: Page Connectivity Graph Construction from the Hotel and Lake Join Index ... | 3 |
| Figure 3: Example of clustering | 6 |
| Figure 4: Example of a CO graph | 7 |
| Figure 5: State of art approach vs. SID approach in refinement step | 16 |
| Figure 6: Example of weighted PCG | 23 |
| Figure 7: Example of Min-Cut Partition | 24 |
| Figure 8: Example of Matrix-based Partition..... | 25 |
| Figure 9: Execution of MST algorithm..... | 28 |
| Figure 10: Execution of Match based algorithm..... | 29 |
| Figure 11: Greedy matching vs. maximum weighed macthing | 31 |
| Figure 12: Algorithm of maximum weighted matching | 32 |
| Figure 13: Example of invalid AMO order..... | 35 |
| Figure 14: A CO graph with ten clusters | 38 |
| Figure 15: Execution of MST algorithm..... | 39 |
| Figure 16: Execution of MBM algorithm | 41 |
| Figure 17: Execution of MWM algorithm..... | 43 |
| Figure 18: Procedure of the study..... | 45 |
| Figure 19: Comparison of total overlapping weight | 49 |
| Figure 20: Comparison of total saving of I/O cost..... | 50 |

List of Tables

| | |
|---|----|
| Table 1: Overview of filtering step approaches | 10 |
| Table 2: Overview of I/O strategies for the refinement step..... | 18 |
| Table 3: Example of sequence candidate pairs returned by filter step..... | 18 |
| Table 4: Execution trace of I strategy | 19 |
| Table 5: Execution trace of Sorting-based strategy | 20 |
| Table 6: Execution trace of segmented strategy | 20 |
| Table 7: Execution trace of Zig-Zag strategy | 21 |
| Table 8: Results of experiment with 10 clusters | 48 |
| Table 9: Summary of experiment results | 50 |

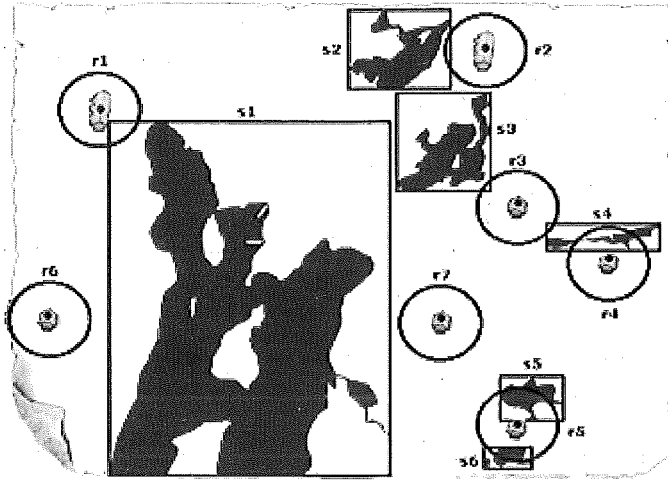
1 Introduction

1.1 The Background to the Study

Today, the use of spatial databases is increasing. These databases are used to manage space-related data. Spatial databases support spatial data types, such as points, lines, polygons and regions. The application of spatial databases can be found in Geographic Information System (GIS), cryptography, image processing, urban planning, geology, astronomy, the study of the human body and molecular structures.

One of the most important query operations in spatial databases is the spatial join, which is the complement to the intersection join in relational database systems (Güting, 1994). The spatial join operation combines two or more sets of spatial objects that satisfy a spatial predicate, mostly *intersection* and *distance within*. The spatial join processing can be space and time expensive due to the large size of the spatial objects and the computationally intensive nature of spatial operations (Xiao, Zhang, & Jia, 2001).

Figure 1 represents two spatial relations: hotel (R) and lake (S). The hotel is represented by a point attribute and the lake is represented by a polygon. An example of a spatial join query can be 'find all hotels within 7 km of a lake'. Figure 1(a) shows the spatial attributes for hotels and lakes. Points $r_1, r_2, r_3, r_4, r_5, r_6,$ and r_7 represent the hotel locations and polygons $s_1, s_2, s_3, s_4, s_5,$ and s_6 represent the Minimum Bounding Rectangle (MBR) for the lake boundaries. The circle around each hotel shows the area within 7 km from the hotel. Figure 1(b) shows hotel and lake relations. The hotel relation has a unique ID, a location, and other non-spatial attributes. The lake relation has its own unique ID, the size of its spatial object, MBR data, a pointer to the actual spatial data, and other non-spatial attributes. To satisfy the query, a join needs to be performed on the hotel and lake relations based on their spatial attributes. To do this, a join index can be used to find the pairs which satisfy the query. A join index is a special data structure that facilitates rapid join query processing and is generally used for data sets that are not updated frequently (Shekhar, Lu, Chawla, & Ravada, 2002). The join index for this join containing the tuple IDs which match the spatial join predicate is shown in Figure 1(c). Each tuple in the join index represents a tuple in the table $JOIN(R, S, distance(R.Location, S.MBR) < 7 \text{ km})$.



(a) Spatial attributes of R and S (adapted from Poirier, 2002)

| ID | Location (x, y) | Non-spatial data |
|----|-----------------|---------------------|
| r1 | (3.4, 8.8) | Name, Nb_Rooms, ... |
| r2 | (7.2, 9.4) | (...) |
| r3 | (8.3, 6.4) | (...) |
| r4 | (9.2, 4.9) | (...) |
| r5 | (8.4, 1.9) | (...) |
| r6 | (0.9, 3.3) | (...) |
| r7 | (6.6, 3.2) | (...) |

| ID | Size (Kb) | MBR | Data Pointer | Non-spatial data |
|----|-----------|--------------------|--------------|-------------------|
| s1 | 456 | (3.5, 0, 6.1, 8.6) | 0x9FFF0 | Name, Region, ... |
| s2 | 44 | (...) | (...) | (...) |
| s3 | 56 | (...) | (...) | (...) |
| s4 | 23 | (...) | (...) | (...) |
| s5 | 27 | (...) | (...) | (...) |
| s6 | 12 | (...) | (...) | (...) |

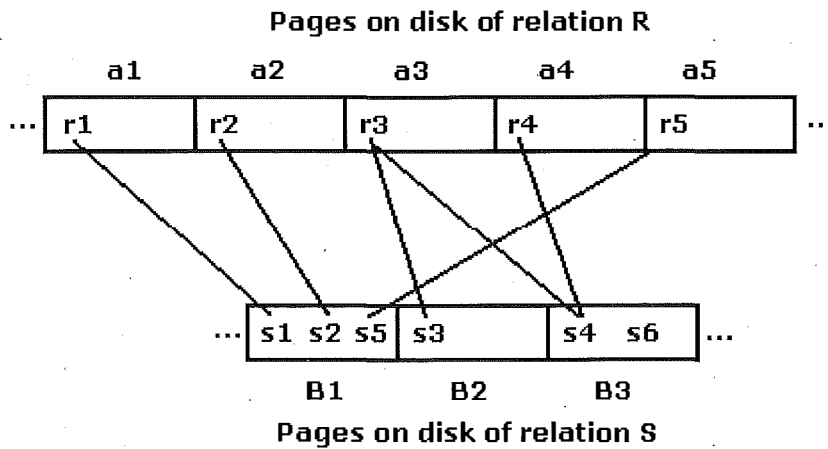
(b) R and S relations (from Poirier, 2002)

| R.ID | S.ID |
|------|------|
| r1 | s1 |
| r2 | s2 |
| r3 | s3 |
| r3 | s4 |
| r4 | s4 |
| r5 | s5 |
| r5 | s6 |

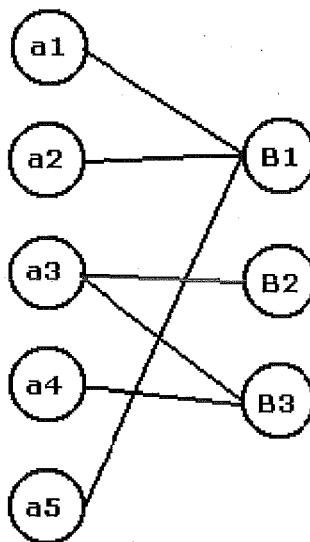
(c) Join index of R and S relations (from Poirier, 2002)

Figure 1: Constructing a join index from two relations

A join index can also be represented by a bipartite graph $G = (V1, V2, E)$, where $V1$ contains the tuple IDs of relation R , and $V2$ contains the tuple IDs of relation S . The edge set E contains an edge (Vr, Vs) for $Vr \in R$ and $Vs \in S$, if there is a tuple corresponding to (Vr, Vs) in the join index. This graph is also known as a Page-Connectivity Graph (PCG) in Shekhar, Lu, Chawla, and Ravada (2002) or Spatial Join (SJ) graph in Xiao, Zhang, and Jia (2001) when the join index between two relations is described at the page level. The PCG for the hotel and lake join index is shown in Figure 2.



(a) Page structure on disk for Join Index



(b) PCG or SJ graph

Figure 2: Page Connectivity Graph Construction from the Hotel and Lake Join Index

Generally, spatial data are stored in secondary storage due to the size of spatial objects. To process a spatial query operation, spatial objects need to be fetched from secondary storage, e.g., hard drive, into main memory for processing. Thus, this process can involve a lot of Input/Output (I/O) cost. It is important to minimise the I/O cost involved in a spatial join operation to maximise the efficiency of the spatial join processing (Xiao, Zhang, & Jia, 2001).

1.1.1 Spatial Join Processing

The majority of spatial join algorithms follow a two-stage process known as filter and refine steps (Orenstein, 1990). The filter step operates on approximations of the actual spatial data, such as MBR. Its purpose is to eliminate most of the candidates that are not relevant for answering the query. The result of this step is a set of potential candidates that are likely to satisfy the spatial predicate. The benefits of the filter step are that it rules out many object pairs that cannot satisfy the spatial predicate without having to fetch them from disk and it is less expensive to operate on approximations of the geometry. Thus, it reduces the overall computation costs (Poirier, 2002).

Due to the approximation for the spatial data, the candidate sets may contain false drops, that is, spatial objects that satisfy the filter operation but do not satisfy the actual spatial predicate. Next, the refinement step consists of fetching the full geometry description of the candidates into memory and eliminating those false drops by performing the spatial operation on the actual geometry. Since the memory size is limited, it can keep only a limited number of spatial objects for computation at a time. As a result, an object could be fetched more than once when it is needed for spatial join operation (Xiao, Zhang, & Jia, 2001). It is essential to schedule the object in a way that fetching is only required once to load the object into memory in order to minimise the I/O cost.

Techniques that aim at minimising the I/O cost can contribute significantly towards reducing the total cost of the spatial join operation. Experiments have shown that disk accesses at the refinement step take a significant amount of time compared to the CPU time required for spatial join (Guttman, 1984). Hence, the best technique to minimise the I/O cost at the refinement step is required.

In spatial join processing, a common method to minimise the I/O cost at the refinement step is known as the housekeeping stage. It consists of two steps, namely: sequencing/scheduling and clustering (Abel, Gaede, Power, & Zhou, 1999). The sequencing step is to sequence the candidate pairs by reducing the number of duplicate fetches before fetching them from disk. The clustering step is to partition the spatial objects into clusters and load them into the main memory cluster by cluster.

1.2 The Significance of the Study

Due to the increasing popularity of spatial databases, researchers have focused their efforts on improving the query processing performance of the most expensive spatial database operation: the spatial join (Güting, 1994). The cost of spatial join processing is usually much more than traditional relational-join processing due to the large sizes of spatial objects and the expensive computation of spatial predicates. The research focuses on finding an efficient algorithm to reduce the I/O cost at the refinement step. This may result in more efficient and faster processing of spatial join operation in spatial databases.

Most approaches perform a housekeeping step between the filtering and refinement steps. The advantage of the housekeeping step is to avoid fetching the same spatial object into memory more than once. Thus, it can minimise the I/O cost at the refinement step. One such novel housekeeping step is a combination of sequencing and clustering steps called cluster-scheduling (Xiao, Zhang, Jia, & Zhou, 2000). First, it partitions spatial objects into clusters such that objects sharing a cluster are closely related and can therefore be brought into memory together for processing. Then, it schedules the cluster loading process in order to minimise the total I/O cost at the refinement step by minimising the number of duplicate fetches. It is essential to schedule the cluster loading process so that two successive clusters in the sequence have maximal number of overlapping objects. However, determining an optimal page access sequence or Maximal Overlapping (MO) order in spatial join processing has been shown to be Non-deterministic Polynomial-time (NP)-complete by Neyer and Widmayer (1997). So, it is required to find a method that is very close to the optimal page access sequence, namely Approximate Maximal Overlapping (AMO) order.

1.3 The Purpose of the Study

According to Shekhar, Ravada, Lu, and Chawla (1998), the cost of spatial join computation, which uses a join-index in a secondary memory, i.e., hard drive, with limited buffer space, depends primarily on the page access sequence used to fetch the pages of the base relations. The determination of an optimal page access sequence such that the join can be computed with the minimum of page re-accesses given a limited number of buffer pages is known as Optimal Page Access Sequence with a Fixed Buffer (OPAS-FB) problem. In spatial join processing, the OPAS-FB problem is significant since the size of spatial data can be large and insufficient main memory capacity can create a bottleneck (Poirier, 2002).

This study focuses at the housekeeping stage, especially at the cluster-scheduling step. In the clustering method, the candidate pairs will be partitioned into several clusters (see Figure 3). The result of the clustering method is a Cluster Overlapping (CO) graph. A CO graph represents the overlapping relationships between data clusters. For example, Figure 4(a) shows the object sizes and Figure 4(b) shows a CO graph corresponding to the clusters in Figure 3(b). The edge weight from cluster V1 to cluster V2 is 250 (60+80+110) as objects B1, B2, and B3 overlap in both clusters.

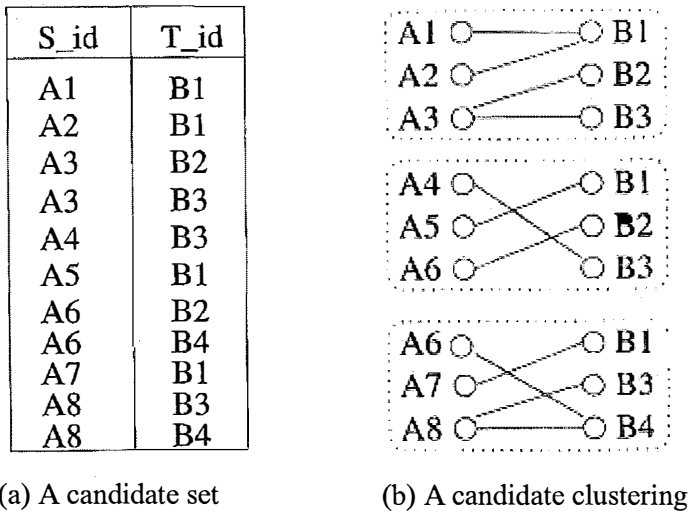
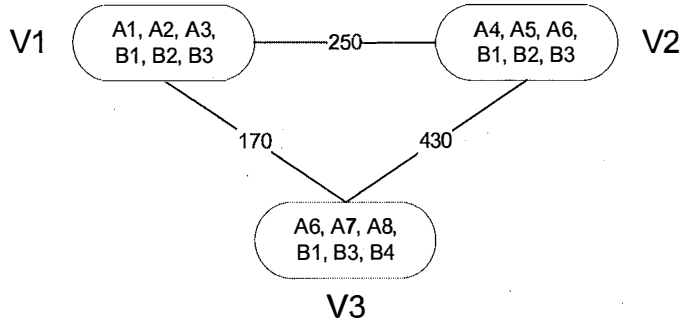


Figure 3: Example of clustering

| Object ID | Object 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) CO graph from Xiao, Zhang, Jia, & Zhou (2000)

Figure 4: Example of a CO graph

The cluster-scheduling method is to schedule the cluster in a way that two consecutive clusters have a maximum number of overlapping objects. Let V_1, V_2, \dots, V_n be the clusters generated by the clustering algorithm. The notion of Maximum Overlapping (MO) order is defined as follows:

Given a CO graph $G = (V, E, w)$ with $V = \{V_1, V_2, \dots, V_n\}$, a MO order among sets V_1, V_2, \dots, V_n is a sequence $(V_{i_1}, V_{i_2}, \dots, V_{i_n})$ such that $\sum_{i=1}^{n-1} size(V_{i_i} \cap V_{i_{i+1}})$ reaches the maximum among all permutations of V .

For example, (V_1, V_2, V_3) is an MO order in the CO graph in Figure 3(a) and the total size of overlapping objects between adjacent nodes in the order is 680. To find an MO order is to check all permutations of V to see which one makes the maximum of $\{\sum_{i=1}^{n-1} size(V_{i_i} \cap V_{i_{i+1}})\}$. Obviously, this method has a factorial order and is not efficient and practical. The research proposes a better algorithm to schedule the sequence of the cluster processing. As a result, the I/O cost at the refinement step will be minimised.

1.4 Statement of Research Questions

Finding the MO order for a CO graph guarantees to generate the best cluster-scheduling sequence. Unfortunately, the problem of finding an MO order in a CO graph has been shown to be NP-complete (Xiao, Zhang, Jia, & Zhou, 2000). The overarching question is to find an Approximate MO (AMO) order that can be used to guide the scheduling of the spatial clusters in spatial join processing. Two areas need to be explored in order to answer the overarching question, namely:

1. Will the use of Maximum Weighted Matching algorithm expedite the finding of AMO order of a CO graph?
2. How well is the AMO order produced by the proposed method compared to other methods, such as: Maximum Spanning Tree (MST) and Match Based Method (MBM) in terms of I/O cost?

2 Review of Relevant Literature

2.1 General Literature Review

According to Huang, Jones, and Rundensteiner (1998), spatial join processing is very expensive in terms of both CPU and I/O costs for three reasons. Firstly, spatial objects are typically represented by structures that require extensive storage. For example, a high-resolution vector representation of a polygon may store thousands of points where each point is represented by an x-coordinate and a y-coordinate value. Secondly, the spatial join operation requires multiple scans of often large data sets. Finally, determining a spatial relation such as the intersection of two objects is very computationally intensive. It requires super-linear time complexity as a function of the number of points used to represent each object.

The first two factors contribute to high I/O costs, whereas the third factor results in high CPU costs. As a result, spatial join queries over large data sets usually incur a long response time. To minimise the CPU and the I/O cost, the spatial join processing usually executes in two steps, namely filter step and refinement step (Orenstein, 1990). These will now be discussed.

2.1.1 Filter Step

Several spatial join algorithms, most of which focus on the filter step, have been proposed, such as: R-tree family (Beckmann, Begel, Schneider, & Seeger, 1990), seeded tree (Lo & Ravishankar, 1994), spatial hash join (Lo & Ravishankar, 1996), and merge join (Patel & DeWitt, 1996). Most work on spatial join processing focuses on the efficient computation of the filter step where MBRs are used for approximating the spatial objects.

Based on the availability of spatial indexes, there are three categories of processing the filter: availability of spatial indexes on both relations participating in the spatial join operation, on one of the relations or on none of the relations (see Table 1). For the first category where spatial indexes have been built on both relations, these indexes are generally used in the implementation of spatial join processing. The second and third category may arise in a complex query where one or both operand sets might be an

interim result from previous operations (Mamoulis & Papadias, 2003). For example, consider a query ‘find all hotels with category 4 stars within 7km of a lake’ and the hotels and lakes are indexed on their spatial extent. If the selection part of the query is performed first before the spatial join, the resulting hotels will be non-indexed. Thus, this is a second category and it requires a single index join algorithm to perform the filtering step.

Table 1: Overview of filtering step approaches

| Filtering Step Approach | |
|-------------------------|---|
| Index on both relations | R-tree family (Beckmann, Begel, Schneider, & Seeger, 1990) |
| | Polygon Map Random (Hoel & Samet, 1992) |
| Index on one relation | INLJ (Mamoulis & Papadias, 2003) |
| | Bulk Loading and Matching |
| | Build and Match Join (Patel & DeWitt, 1996) |
| | Sort and Match (Papadopoulos, Rigaux, & Scholl, 1999) |
| | Seeded Tree (Lo & Ravishankar, 1994) |
| | Slot Index Spatial Join (Mamoulis & Papadias, 2003) |
| No index | Plane Sweeping (Arge, Procopiuc, Ramaswamy, Suel, & Vitter, 1998) |
| | Spatial Hash Join (Lo & Ravishankar, 1996) |
| | PBSM (Patel & DeWitt, 1996) |
| | S3J (Koudas & Sevcik, 1997) |
| | SSSJ (Arge, Procopiuc, Ramaswamy, Suel, & Vitter, 1998) |

Indexes on both relations

One of the most widely used spatial index structures is R-tree family. There are several varieties of R-tree family in literature, such as R-tree (Guttman, 1984), R^+ -tree (Sellis, Roussopoulos, & Faloutsos, 1987), and R^* -tree (Beckmann, Begel, Schneider, & Seeger, 1990). According to Beckmann, Begel, Schneider, and Seeger (1990), the most efficient of R-tree family is R^* -tree. Brinkhoff, Kriegel, and Seeger (1993) presented a spatial join algorithm where each of the relations is indexed by an R^* -tree. This algorithm synchronously traverses both trees and joins all pairs of overlapping regions. It is based on depth-first traversal of R-trees. This join method is considered as one of the most important ones due to its efficiency and the availability of R-trees in advanced database management systems.

A different traversal strategy was presented by Huang, Jing, and Rundensteiner (1997) which is superior to the depth-first traversal of Brinkhoff, Kriegel, and Seeger (1993) when a large buffer is available. Hoel and Samet (1992) propose the use of Polygonal Map Random (PMR) quadtrees for the spatial join and compare it against members of the R-tree family. In subsequent work, they considered the problem of spatial joins

when Point Region (PR) quadtrees are on the input relations (Hoel & Samet, 1995). All of the proposed techniques are to speed up the filter step of the spatial join.

Index on one relation

The simplest method for the case that only one input is indexed is the Indexed Nested Loop Join (INLJ). In accordance with its relational join counterpart, INLJ applies a window query to the R-tree for every object in the non-indexed data set (Mamoulis & Papadias, 2003). The build and match join (Patel & DeWitt, 1996) builds an R-tree from the raw input using bulk loading and joins it with the existing tree using R-tree Join. Sort and match (Papadopoulos, Rigaux, & Scholl, 1999) employs the Sort-Tile-Recursive (STR) technique of Leutenegger, Edgington, and Lopez (1997) to sort the rectangles from the non indexed input but, instead of building the packed tree, it directly matches in memory created leaf nodes. For each produced leaf node, a window query is executed and plane sweep is applied to join it with all leaf nodes from the existing R-tree that intersect it.

Lo and Ravishankar (1994) discussed the case where exactly one of the relations does not have an index. They proposed a method called Seeded Tree, which makes use of an R-tree index already available on one data set to construct dynamically an index for the second data set at the join time. Their work on seeded trees was the first that addressed the problem of processing spatial joins when only one R-tree is available. Seeded trees are R-tree-like structures, and are divided into the seed levels and the grown levels. The nodes in the seed levels are used to guide tree growth during tree construction. The seed levels can also be used to filter out some input data during construction, thereby reducing tree size. They developed a technique that uses intermediate linked lists during tree construction and significantly speeds up the tree construction process. Once the index is constructed, the tree join algorithm of Brinkhoff, Kriegel, and Seeger (1993) is used to perform the actual join.

Mamoulis and Papadias (2003) investigated whether existing single-index join algorithms have certain limitations. For instance, the INLJ, which applies a window query to the R-tree for each object in the non-indexed set, can be very expensive in terms of both I/O and computational cost. The seeded tree join of Lo and Ravishankar (1994), which creates an R-tree for the non-indexed data, is not appropriate in many cases because of its prohibitive I/O cost. Methods like bulk loading and matching,

sorting and matching apply external sorting on the non-indexed data and totally or partially build an on-the-fly R-tree in order to join it with the existing one. Therefore, these methods have a disadvantage in cases where the non-indexed input is an intermediate result of an underlying operator because they need to materialize it before processing it.

An improvement of the above algorithm has been suggested in Mamoulis and Papadias (2003) where the available main memory is exploited more efficiently. They propose Slot Index Spatial Join (SISJ), a hash join algorithm that overcomes most of the above deficiencies. SISJ distributes the R-tree entries at a specific level into S partitions, called slots, and builds an in-memory index from them. The slot index keeps for each slot the identifiers of the nodes pointed to by the corresponding entries along with the MBR of the entries. All data from the non-indexed input are hashed into buckets with same extents as the slot MBRs. The hash-buckets are finally joined with the R-tree data under the corresponding slot.

Additionally, Mamoulis and Papadias (2003) present two I/O and CPU optimisation methods that are applied in the join phase of SISJ and significantly improve its performance, namely a bucket ordering heuristic and a repartitioning heuristic. A bucket ordering heuristic joins first the hash buckets with a few pages on disk in order to avoid writing and reading again their in-memory parts. This technique reduces the number of page accesses, especially when the non-indexed input is only slightly larger than the available memory. A repartitioning heuristic improves the computational performance of the algorithm and further reduces its space requirements. After the application of these optimization methods, the overall cost of SISJ drops about 35 percent compared to the initial implementation (Mamoulis & Papadias, 2003).

No indices

The problem of join processing also has been examined under the assumption that no index is available. There are many efficient algorithms for computing the spatial join of two non-indexed spatial data sets in the case where both sets fit in main memory. One example, which derives from computational geometry, is the Plane Sweeping (Arge, Procopiuc, Ramaswamy, Suel, & Vitter, 1998). It is used to determine if the spatial relation specified in the join query exists between two spatial objects by

using rectangle intersection. A plane-sweeping algorithm for rectangle intersection only has to find all intersections between rectangles located on the same sweepline.

Two algorithms have been proposed for the case where the data sets do not fit in main memory. Lo and Ravishankar (1996) introduced Spatial Hash Join (SHJ) and Patel and DeWitt (1996) proposed Partition Based Spatial Merge Join (PBSM). Both methods divide the datasets into smaller partitions, such that each partition fits in memory, and apply a join algorithm to each pair of partitions. PBSM replicates some of the data of both input relations to improve join processing, whereas the spatial-hash join only allows replication on one relation. They both introduce replication of the entities in partitions in order to compute the join. However, the replication can result in poor performance of the spatial join processing.

Prompted by the above problem, Koudas and Sevcik (1997) present an alternative algorithm that requires no replication. They show the benefits of avoiding replication in such cases. They introduced a method without data replication, called Size Separation Spatial Join (S3J). S3J imposes a dynamic hierarchical decomposition of the space and permits an efficient joining phase. The Dynamic Spatial Bitmap feature of S3J can be implemented using bitmap indexing techniques already available in most relational systems. They presented an analytical and experimental comparison of S3J with PBSM and SHJ algorithms for computing spatial joins when indices do not exist for the data sets involved. Using a combination of analytical techniques and experimentation with real and synthetic data sets, they showed that S3J outperforms these two methods for a variety of types of spatial data sets.

Arge, Procopiuc, Ramaswamy, Suel, and Vitter (1998) proposed another approach without data replication, called Scalable Sweeping-Based Join (SSSJ). SSSJ is an external sweep-line algorithm which tries to keep the status of the sweep-line in main memory. SSSJ achieves both efficiency on real-life data and robustness against highly skewed and worst-case data sets on both internal computation time and I/O transfer. They present experimental results based on an efficient implementation of the SSSJ algorithm, and compare it to the original as well as an optimised PBSM algorithm of Patel and DeWitt (1996). However, a serious deficiency of SSSJ is that both input relations have to be sorted first before producing the initial output tuple.

Dittrich and Seeger (2000) proposed several improvements of PBSM (Patel & DeWitt, 1996) and S3J (Lo & Ravishankar, 1996), particularly on the impact of data redundancy and duplicate detection on the performance of these methods. For PBSM, they present a simple and inexpensive online method to detect duplicates in the response set. There is no need to eliminate duplicates in a final sorting phase as was suggested originally. They also investigate the impact of different internal algorithms on the total runtime of PBSM.

S3J has been proposed as an algorithm that avoids the problem of duplicates in the response set by simply avoiding the generation of redundant data objects. However, Dittrich and Seeger (2000) show that data redundancy results in substantial performance improvements for S3J. They introduce replication of data objects and show that the total processing cost can be reduced considerably. Duplicates in the response set can be detected at very little cost using a slightly modified version of the method suggested for PBSM. Moreover, they also address the problem of choosing an efficient internal algorithm for S3J (Dittrich & Seeger, 2000). Results of a large set of experiments with real data sets reveal that Dittrich and Seeger (2000) suggested modifications of PBSM and S3J result in substantial performance improvements where PBSM is generally superior to S3J.

2.1.2 Refinement Step

The refinement step can be separated into two parts:

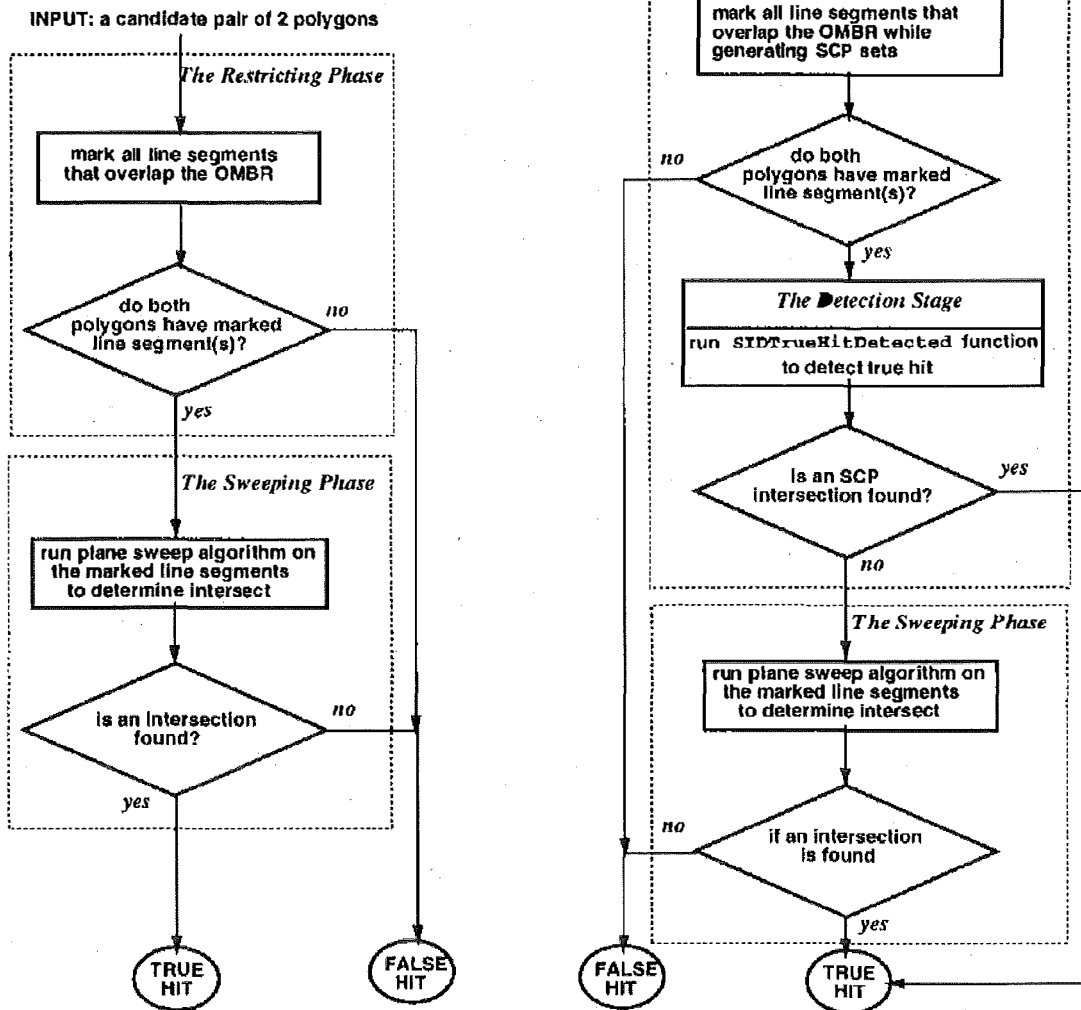
1. Fetching full geometry description of the candidates, which are produced by the filter step, into memory, and
2. Performing the spatial join operation on the actual geometry.

The first part implies high I/O cost since the size of spatial objects to be fetched from disk to memory can be very large. The second part implies high CPU cost as the spatial operations are computationally very intensive.

After retrieving from the candidate set a polygon-pair and its vector representations, the refinement step determines the intersect relation between the two polygons by processing the restricting phase and the sweeping phase (Huang, Jones, & Rundensteiner, 1998).

Huang, Jones, and Rundensteiner (1998) proposed a new technique focusing on the optimisation of the refinement step. They propose a screen-test procedure to be executed before the plane sweep algorithm that substantially reduces the computation required during refinement. They call this procedure Symbolic Intersect Detection (SID).

Figure 5 shows SID performs efficient true hit detection in two stages, namely clipping stage and detection stage, to replace the restriction phase in the original refinement step. In the clipping stage, to determine if two candidate polygons intersect, SID uses their Overlapping MBR (OMBR) to clip all segments of the two polygons which overlap the OMBR. SID abstracts each segment by a compact symbolic representation using only offsets of the sides of the OMBR. In the detection stage, based on this abstract information, SID efficiently detects situations under which two clipped segments cross each other deterministically. When such a crossing is determined between two clipped segments, their association polygons therefore are guaranteed to intersect. As a result, performance is improved because further intersect computation such as the computationally intensive plane sweep algorithm is not needed for the true hit candidates detected by SID (Huang, Jones, & Rundensteiner, 1998).



(a) The state of art of refinement step (b) Refinement step with SID optimisation

Figure 5: State of art approach vs. SID approach in refinement step

Additionally, they present an analytical cost model characterising SID's effectiveness under various conditions. Based on real map data, they also run experiments comparing the performance of the state-of-the-art spatial join approach with the spatial join using the SID optimisation. The results show that their SID optimisation effectively detects more than 80% of the true hits with negligible overhead. Consequently, with the SID optimisation, the time intersect computation in the refinement step is improved by over 50%, as predicted by the analytical model (Huang, Jones, & Rundensteiner, 1998).

2.2 Literature on Previous Findings

According to Abel, Gaede, Power, and Zhou (1999), different filter algorithms and the presence of spatial indexes generate significant variations in the ordering of the set of candidate pairs generated by the filtering step. To solve this problem, Patel and DeWitt (1996) introduced a housekeeping step in between filter and refinement steps. The housekeeping step performs sequencing of the candidate pairs before fetching them from disk. It aims at minimising the I/O cost involved in the refinement phase by reducing the number of duplicate fetches.

To minimise the overall I/O cost at the refinement step, most of techniques will provide a housekeeping step. The housekeeping step is generally divided into two methods, namely sequencing/scheduling and clustering.

Abel, Gaede, Power, and Zhou (1999) classified two types of refinement strategies, namely: immediate and deferred processing. In immediate processing, candidate pairs are tested in the refinement step as they are generated by the filter step. In this strategy, no sequencing takes place during the housekeeping step. On the other hand, the deferred processing orders the candidate pairs before fetching them from disk. The full set of candidate pairs is assembled in the housekeeping step before applying the refinement step.

Several heuristic solutions have been proposed for the OPAS-FB problem. They can be divided into two categories, namely asymmetric and symmetric methods. An asymmetric method favours objects from one relation, i.e., sorting the join-index on one of the join keys, whereas a symmetric method has no preferences. An asymmetric strategy can be advantageous if one set is much larger than the other or if the objects of one data set are well clustered spatially (Poirier, 2002).

Table 2 shows varieties of housekeeping approaches that aim to minimise the I/O cost for the refinement step.

Table 2: Overview of I/O strategies for the refinement step

| IMMEDIATE | DEFERRED | | |
|----------------|---|---|---------------------------------------|
| | Asymmetric | Symmetric | |
| Naïve Strategy | Sorting-based Heuristic (Valduriez, 1987) | Travelling Salesman Heuristic | Scheduling & No Clustering |
| | Segmented Strategy (Patel & DeWitt, 1996) | Zig-Zag Strategy (Abel, Gaede, Power, & Zhou, 1999) | |
| | | Greedy Heuristics: - FPH (Fotouhi & Pramanik, 1989) - OH (Omiecinski, 1989) - COH (Chan & Ooi, 1997) | |
| | AGP (Shekhar, Lu, Chawla, & Ravada, 2002) | SGP (Shekhar, Lu, Chawla, & Ravada, 2002) | Clustering & Scheduling |
| | | Matrix Permutation (Xiao, Zhang, & Jia, 2001) | |
| | | MST (Xiao, Zhang, Jia, & Zhou, 2000) | |
| | | MBM (Xiao, 2003) | |

2.2.1 Sequencing/Scheduling Strategies

The next few sections will use Table 3. It shows an example of sequence candidate pairs returned by the filter step (Abel, Gaede, Power, & Zhou, 1999). Using this example, suppose the buffer size is 4 and all spatial objects are the same size, namely 1, and the cache is initially empty.

Table 3: Example of sequence candidate pairs returned by filter step

| | |
|----|----------|
| 1 | (r6, s3) |
| 2 | (r7, s1) |
| 3 | (r2, s1) |
| 4 | (r1, s2) |
| 5 | (r2, s6) |
| 6 | (r1, s3) |
| 7 | (r2, s7) |
| 8 | (r3, s1) |
| 9 | (r4, s1) |
| 10 | (r5, s2) |
| 11 | (r2, s5) |
| 12 | (r1, s1) |
| 13 | (r2, s4) |

Naïve Strategy

The naïve strategy processes candidate pairs in the order generated by the filter algorithm. No sequencing is performed in a housekeeping step. For each candidate pair, the spatial descriptions of objects are fetched into memory. It is assumed that the spatial descriptions of any given pair of objects will fit into memory in order to perform the computational geometry algorithm on them. Once the buffer is full, the replacement policy used is least-recently-used (Poirier, 2002). To use the candidate pairs from Table 3, the naïve strategy requires 20 load requests (see Table 4).

Table 4: Execution trace of I strategy

| Candidate Pair | Cache Content | Number of Load Requests |
|---------------------------------------|------------------|-------------------------|
| (r6, s3) | (r6, s3) | 2 |
| (r7, s1) | (r6, s3, r7, s1) | 2 |
| (r2, s1) | (s3, r7, s1, r2) | 1 |
| (r1, s2) | (s1, r2, r1, s2) | 2 |
| (r2, s6) | (r1, s2, r2, s6) | 1 |
| (r1, s3) | (r2, s6, r1, s3) | 1 |
| (r2, s7) | (r1, s3, r2, s7) | 1 |
| (r3, s1) | (r2, s7, r3, s1) | 2 |
| (r4, s1) | (s7, r3, s1, r4) | 1 |
| (r5, s2) | (s1, r4, r5, s2) | 2 |
| (r2, s5) | (r5, s2, r2, s5) | 2 |
| (r1, s1) | (r2, s5, r1, s1) | 2 |
| (r2, s4) | (r1, s1, r2, s4) | 1 |
| Total number of load requests: | | 20 |

Sorting-Based Strategy

This strategy is also known as the ‘Simple Strategy’ in (Abel, Gaede, Power, & Zhou, 1999). The sorting-based strategy, one of the deferred processing strategies, is to defer the refinement step until all candidate pairs are available. The list of candidate pairs is sorted according to the object identifiers of one relation. The list is then processed sequentially and ensures that there are no duplicate fetches for objects of the first relation. Clearly, some objects of the second relation will have to be fetched more than once (Poirier, 2002). Table 5 shows the sorting-based strategy outperforms the naïve strategy with a total number of 17 load requests.

Table 5: Execution trace of Sorting-based strategy

| Candidate Pair | Cache Content | Number of Load Requests |
|---------------------------------------|------------------|-------------------------|
| (r1, s1) | (r1, s1) | 2 |
| (r1, s2) | (r1, s1, s2) | 1 |
| (r1, s3) | (r1, s1, s2, s3) | 1 |
| (r2, s1) | (r2, s2, s3, s1) | 1 |
| (r2, s4) | (r2, s3, s1, s4) | 1 |
| (r2, s5) | (r2, s1, s4, s5) | 1 |
| (r2, s6) | (r2, s4, s5, s6) | 1 |
| (r2, s7) | (r2, s5, s6, s7) | 1 |
| (r3, s1) | (r3, s6, s7, s1) | 2 |
| (r4, s1) | (r4, s6, s7, s1) | 1 |
| (r5, s2) | (r5, s7, s1, s2) | 2 |
| (r6, s3) | (r6, s1, s2, s3) | 2 |
| (r7, s1) | (r7, s2, s3, s1) | 1 |
| Total number of load requests: | | 17 |

Segmented Strategy

Valduriez (1987) introduced an efficient sequencing algorithm for join processing using join indices and it was adapted by Patel and DeWitt (1996) for spatial joins. A segment is composed of a maximum number m of the first relation objects. Segmented sequencing is an asymmetrical strategy. It guarantees that no object in R within a segment is loaded twice. However, objects in S appearing in different segments are likely to be fetched multiple times (Poirier, 2002). For the example in Table 3, the segmented strategy requires 18 load requests (see Table 6).

Table 6: Execution trace of segmented strategy

| Candidate Pair | Cache Content | Number of Load Requests |
|---------------------------------------|------------------|-------------------------|
| (r1, s1) | (r1, r2, r3, s1) | 4 |
| (r2, s1) | (r1, r2, r3, s1) | 0 |
| (r3, s1) | (r1, r2, r3, s1) | 0 |
| (r1, s2) | (r1, r2, r3, s2) | 1 |
| (r1, s3) | (r1, r2, r3, s3) | 1 |
| (r2, s4) | (r1, r2, r3, s4) | 1 |
| (r2, s5) | (r1, r2, r3, s5) | 1 |
| (r2, s6) | (r1, r2, r3, s6) | 1 |
| (r2, s7) | (r1, r2, r3, s7) | 1 |
| Load Next Segment | | |
| (r4, s1) | (r4, r5, r6, s1) | 4 |
| (r5, s2) | (r4, r5, r6, s2) | 1 |
| (r6, s3) | (r4, r5, r6, s3) | 1 |
| Load Next Segment | | |
| (r7, s1) | (r4, r5, r7, s1) | 2 |
| Total number of load requests: | | 18 |

Zig-Zag Strategy

The zig-zag strategy was proposed by Abel, Gaede, Power, and Zhou (1999). It is a symmetric strategy. It dynamically alternates between the two data sets. This approach is expected to perform best when the data sets to be joined have a similar data distribution. The algorithm maintains two lists, F and F' . The F list contains candidate pairs sorted by r_i and the F' list contains the candidate pairs sorted by s_j . If all cache objects of one set are processed, the algorithm switches to the respective other set and processes all cache objects of this set. For the implementation, it is necessary to extend each entry $e_k = (r_i, s_j) \in F$ with a flag indicating whether this pair has been processed or not. This flag is necessary since in the course of zig-zagging some pairs may be skipped and will have to be processed later. Also, two cursors have to be maintained along with a list reflecting the current content of the cache. Table 7 shows the total number of load request for Zig-Zag strategy is 17.

Table 7: Execution trace of Zig-Zag strategy

| Candidate Pair | Cache Content | Number of Load Requests |
|---------------------------------------|------------------|-------------------------|
| (r1, s1) | (r1, s1) | 2 |
| (r1, s2) | (s1, r1, s2) | 1 |
| (r1, s3) | (s1, s2, r1, s3) | 1 |
| R completely processed, go to S | | |
| (r2, s1) | (s2, s3, s1, r2) | 1 |
| (r3, s1) | (s3, r2, s1, r3) | 1 |
| (r4, s1) | (r2, r3, s1, r4) | 1 |
| (r7, s1) | (r3, r4, s1, r7) | 1 |
| S completely processed, go to R | | |
| (r2, s4) | (r4, r7, r2, s4) | 2 |
| (r2, s5) | (r7, s4, r2, s5) | 1 |
| (r2, s6) | (s4, s5, r2, s6) | 1 |
| (r2, s7) | (s5, s6, r2, s7) | 1 |
| R completely processed, go to S | | |
| (r5, s2) | (s6, s7, s2, r5) | 2 |
| (r6, s3) | (s2, r5, s3, r6) | 2 |
| Total number of load requests: | | 17 |

Scheduling

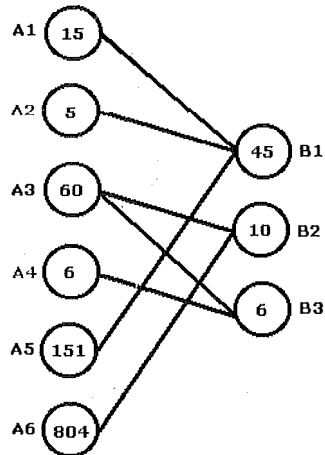
Lim, Pheng, and Chong (2001) researched the problem of scheduling page accesses in database join processing. Their research questions focus on two interesting problems. First, they determine a page access sequence that uses the minimum number of buffer pages without any page being fetched more than once. Second, they determine a page access sequence that minimises the number of page re-accesses for a given buffer size. They use a graph model to represent pages from the relations that contain tuples to be

joined and present a new heuristic for the two problems. This new heuristic is based on the concept of finding the best *N-release-K* sequence at each iteration. They use the notion *N-release-K* to denote *N* pages are brought into the buffer to release *K* pages. Each page access sequence is uniquely expressed as a sequence of segments, where each instance of a page release marks the end of a segment. While the sequence of pages in a segment is fetched, no pages in the buffer can be made available until the last page in the segment is fetched. This is the basis of their new heuristic. Their experimental results show that their new heuristic performs well, especially when comparing to Chan and Ooi Heuristic (COH) (Chan & Ooi, 1997).

Spatially-Augmented Greedy Heuristic (SAGH)

Poirier (2002) proposed a technique called Spatially-Augmented Greedy Heuristic (SAGH). The method efficiently produces a page access sequence that results in a good buffer utilisation as well as disk I/O cost for spatial join processing of non-uniform sized spatial objects. He introduced the method based on the traditional relational join greedy heuristics but it accounts for the size of spatial objects. Some of the traditional relational join greedy heuristics are Fotouhi and Pramanik Heuristic (FPH) (Fotouhi & Pramanik, 1989), Omiecinski Heuristic (OH) (Omiecinski, 1989), and COH (Chan & Ooi, 1997). These greedy heuristics are symmetric and do not require any sorting of the join-keys of either relation. They select the next page or the next set of pages to be fetched into memory based on the pages already in memory and the remaining edges to be processed in a PCG. The selection is often based on the number of neighbours in memory and the number of neighbours on disk. The greedy heuristics generally perform well, are easy to implement, and do not require any pre-processing like sorting or clustering.

| V | w(V) |
|----|------|
| A1 | 15 |
| A2 | 560 |
| A3 | 6 |
| A4 | 151 |
| A5 | 804 |
| A6 | 45 |
| B1 | 10 |
| B2 | 6 |
| B3 | 6 |



(a) Nodes and their weights

(b) a PCG (from Xiao, Zhang, & Jia, 2001)

Figure 6: Example of weighted PCG

In the SAGH, each node of the PCG will be assigned a weight, which corresponds to the size of each spatial object, i.e. weighted PCG = (V, E, w). Figure 6 shows an example of weighted PCG with the weight for each node. According to Poirier (2002), the SAGH chooses $r_i \in R$ and $s_j \in S$ from the PCG to load in the buffer such that:

- (r_i, s_j) is connected
- The sum of the degree of r_i and s_j is minimal
- In case of a tie, select r_i and s_j with the smallest combined size
- (r_i, s_j) not processed yet

Then, the buffer is added with the new node p using the following strategy:

- Find a node q in the buffer who has the smallest non-resident degree (but not zero)
- If there is none, use load policy described above
- Find the node p such that:
 - (q, p) is connected and not processed;
 - p has the smallest non-resident degree;
 - p fits in memory; and then
 - resolves tie by selecting p with smallest size
- If p does not fit in memory, go to next smallest non-resident degree and repeat same procedure
- If no neighbour of q fits in memory, select smallest non-resident degree, even though it does not fit in memory
- Resolve tie by selecting p with smallest size

If a node in memory has to be replaced, then choose the node that:

- has the smallest non-resident degree;
- is not connected to the new node; and then
- resolves ties by expelling the node with the largest size.

Poirier (2002) proved that his SAGH method can compete with the clustering method of Xiao, Zhang, and Jia (2001) by indirectly comparing the SAGH with the Naïve strategy and the Sorting-based heuristic. Moreover, the SAGH heuristic has small pre-processing requirements compared to clustering methods where objects have to be clustered.

2.2.2 Clustering Strategies

Shekhar, Lu, Ravada, and Chawla (1998) introduced an off-line spatial clustering technique based on min-cut graph partitioning of the bipartite PCG for the join index. The technique can minimise the length of the page access sequence, given a fixed buffer size. Consequently, I/O cost can be reduced. A min-cut node partition of a graph $G = (V, E)$ partitions the nodes in V into disjoint subsets while minimising the number of edges in the cut-set. The cut-set of a min-cut partition is the set of edges whose incident nodes are in two different partitions. Figure 7 shows an example of PCG with its corresponding min-cut partition. In this schema, an optimal I/O page access sequence would be obtained if all node clusters were edge disjoint, i.e. the cut-set is empty, and assuming that each cluster can fit into memory.

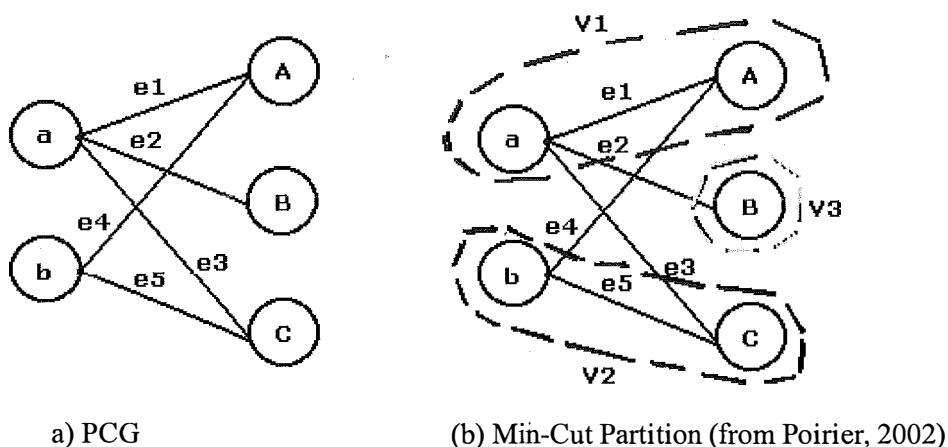


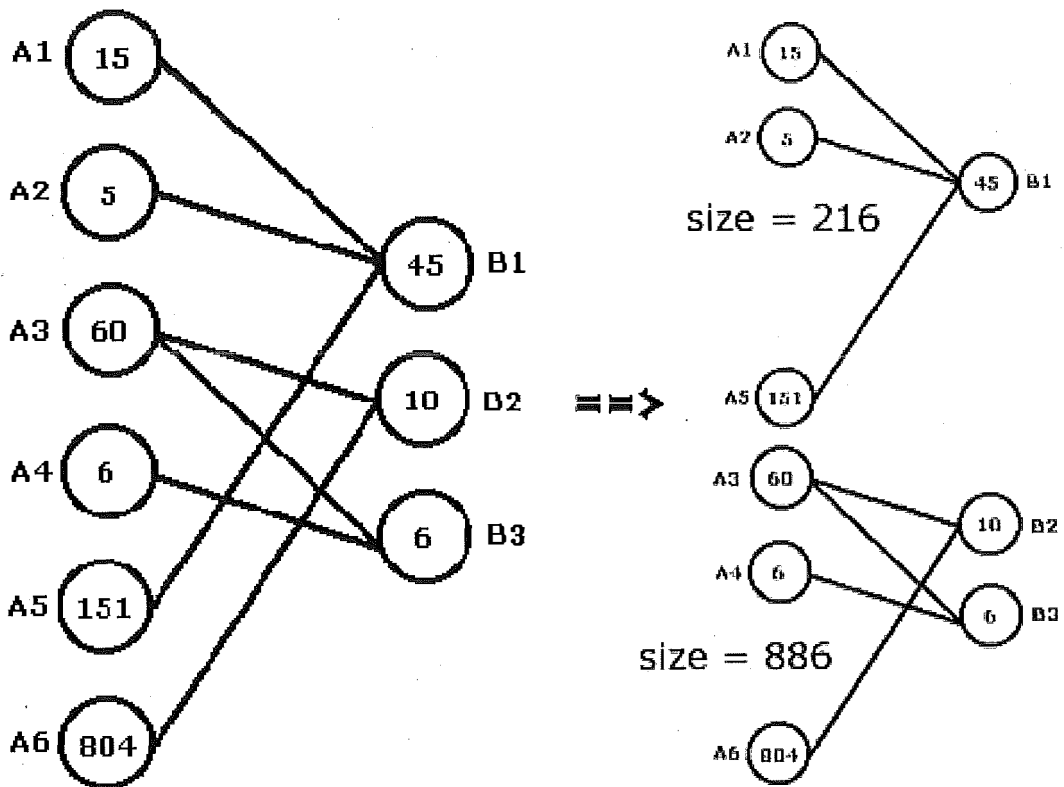
Figure 7: Example of Min-Cut Partition

They used datasets from the Sequoia 2000 project (Stonebraker, Frew, & Dozier, 1993) to do their experiments. The result of the experiments shows that the graph partitioning method outperforms the methods based on sorting and greedy heuristics (Shekhar, Lu, Ravada, & Chawla, 1998). However, it is not clear whether their method really

outperforms the greedy heuristics since the pre-processing time is not accounted in their experiments. The greedy heuristics have no pre-processing step whereas their method has to convert PCG into a hyper-graph and perform a min-cut partitioning of this hyper-graph.

Xiao, Zhang, and Jia (2001) proposed a clustering method that aims to minimise the I/O cost at the refinement stage of spatial join processing. They formalise the problem by using a graph model. Then, from the graph model a matrix-based algorithm is developed to cluster objects such that the objects in the same cluster are closely related. Their method is derived from the Bond Energy Algorithm (BEA) and then they modify it for the candidate-cluster purpose. Similar to the graph partitioning approach, the candidate pairs in their approach are clustered into disjoint sets before each cluster is scheduled in an efficient manner. However, instead of using a graph partitioning method, they use a matrix permutation and decomposition heuristic. The sum of all spatial objects inside a cluster has to be smaller or equal to the buffer size.

Figure 8 shows an example of how the matrix-based method partitions the weighted PCG with the buffer size 1024 bytes.



(a) Weighted PCG (from Xiao, Zhang, & Jia, 2001) (b) Clustering with buffer = 1024

Figure 8: Example of Matrix-based Partition

From their experiments, their method can save 20% and 35% of I/O cost to the case of sorting method and no clustering method respectively (Xiao, Zhang, & Jia, 2001). However, it would be beneficial to perform the experiments using data sets from real GIS applications instead of generated artificial data for more reliable results and acceptability by practitioners in the industry. Today, there are already other methods that can save more I/O cost. However, many of them use this clustering method in addition to their proposed methods.

Shekhar, Lu, Chawla, and Ravada (2002) introduced two new heuristics to solve the OPAS-FB problem. The first one is the Asymmetric Graph Partitioning (AGP) method. AGP method is an improvement over a sorting-based method for spatial join. However, its buffer can be poor since it gives almost the entire buffer space to one relation. The second one is Symmetric Graph Partitioning (SGP) method which uses clustering for the pages of both relations. Both AGP and SGP methods rely on min-cut graph partitioning of the PCG, a technique they proposed previously in Shekhar, Lu, Ravada, and Chawla (1998). The difference between the SGP and AGP methods is that SGP method clusters pages from both tables with no preference to either one. Thus, SGP method minimises the page accesses on both relations.

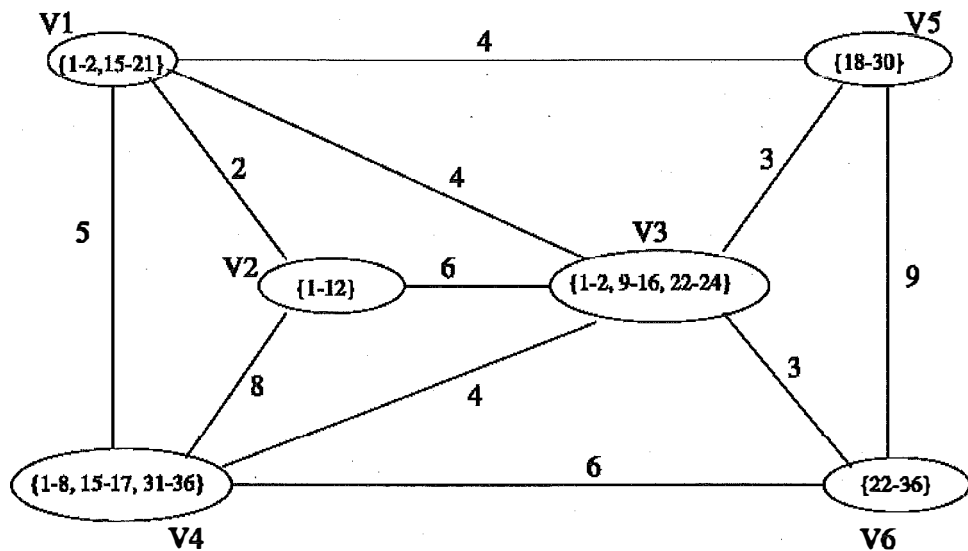
Once again, they did their experiments using real map datasets from the Sequoia 2000 project (Stonebraker, Frew, & Dozier, 1993). They compare their SGP method with the greedy heuristics, namely FPH, OH, and COH, for symmetric processing regarding buffer size, page size, and edge ratio. The result of their experiments shows that SGP method outperforms the greedy heuristics when the memory size is relatively small (Shekhar, Lu, Chawla, & Ravada, 2002). However, since this new method is relied on min-cut graph partitioning, it is not clear whether the SGP method really outperforms the greedy heuristics since the pre-processing time for graph partitioning is not accounted in their experiments.

2.3 Specific Studies Similar to the Current Study

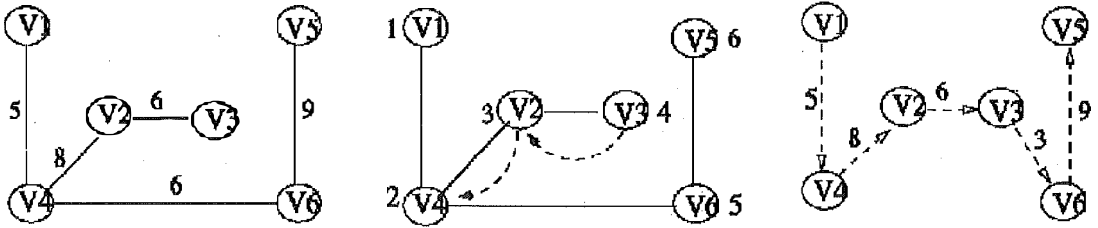
Xiao, Zhang, Jia, and Zhou (2000) proposed a new graph-based cluster-scheduling technique called Maximum Spanning Tree (MST). The idea of this method is to schedule the resulting clusters, which have been generated by the clustering techniques (Shekhar, Lu, Ravada, & Chawla, 1998; Xiao, Zhang, & Jia, 2001), in a way that two consecutive clusters in the sequence have maximal number of overlapping objects. Thus, there is no need to load those overlapping objects when processing the next cluster since they are already in the memory. Consequently, the I/O cost can be minimised, this being the key issue for their cluster-scheduling technique.

The MST algorithm is used to produce an AMO order. The algorithm consists of three steps: find a maximum weight spanning tree T of the CO graph, conduct a depth-first search (DFS) on T , and construct an AMO order according to the output of previous step. The complexity of the MST algorithm is $O(m^2 \log_2 m)$, where $m = \max(|V|, |E|)$. From their simulation, they found that if the spatial join operations are processed cluster by cluster according to the sequences produced by their algorithm, over 50% of the fetching time used for fetching those overlapping objects can be saved.

Figure 9 shows an example of how the MST method works. In this example, the MST method has an AMO order with total overlapping weight of 31 (this will be compared with the next method).



(a) A CO graph (from Xiao, Zhang, Jia, & Zhou, 2000)



(b) A maximum spanning tree (c) A depth-first search of T (d) Approximation to MO

Figure 9: Execution of MST algorithm

Xiao (2003) introduced a new efficient method that produces a better sequence of AMO than the original algorithm, MST, he proposed previously (Xiao, Zhang, Jia, & Zhou, 2000). The new method is called a match based method (MBM). The MBM pseudo-code is:

Algorithm: matchBasedAMO(G)

Input: $G = (V, E, w)$; a CO graph with $V = \{V_1, V_2, V_3, \dots, V_n\}$

Output: $V_{i1}, V_{i2}, \dots, V_{in}$; AMO order of G

Begin

1. Find a maximal match M of G using Greedy matching algorithm;
2. If no matching was found, return;
3. Add all matching into AMO;
4. Coarsen G by collapsing matching nodes of M to produce a coarser graph G' ;
5. matchBasedAMO(G');
6. return;

End

The MBM method uses a greedy matching algorithm to find the maximum weight matchings in a CO graph. The greedy matching algorithm is an approximation algorithm for solving the weighted matching problem and has a performance ratio of $\frac{1}{2}$ to optimal matching (Doratha & Hougardy, 2003). It can be implemented with a running time $O(|E| \log |V|)$ if the edges of G are sorted in a pre-processing step by decreasing weight.

The pseudo-code for greedy matching algorithm:

Greedy Matching ($G = (V, E, w)$)

$M = \Phi$

While $E \neq \emptyset$ do begin

 let e be the heaviest edge in E

 add e to M

 remove e and all edges incident to e from E

End

Simulations have been conducted to demonstrate the saving of I/O cost in spatial join by using the match based algorithm. The result shows that over 67% of the fetching time used for fetching those overlapping objects can be saved.

Use of the same CO graph in Figure 9(a), this MBM produces an AMO order with a total overlapping weight of 33, which is a better result than the MST method (see Figure 10).

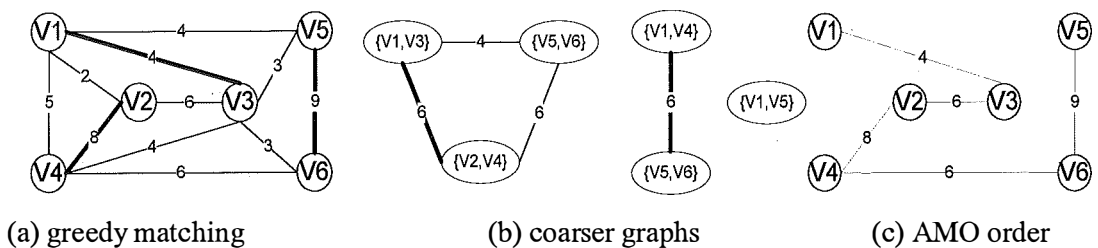


Figure 10: Execution of Match based algorithm

Xiao (2003) presents his algorithm and compares it with three other methods: overlapping-free scheduling (OFS), random-overlapping scheduling (ROS), and the MST. The OFS method fetches objects, cluster by cluster, into memory for the join operations. After a cluster of objects are processed, the data in the memory are cleared before the next cluster is fetched. The ROS method fetches objects, cluster by cluster, in a random way, that is, a cluster is selected randomly. If the objects in the cluster are not in the memory, they are fetched into memory. The experiment results are reliable as he runs the experiments ten times and takes the average of the results. However, most of the datasets in his experiments are generated artificially and only a small proportion of datasets are from real applications.

3 Theoretical Framework

3.1 Identification of Variables Impacting On the Research Questions

There are several variables that impact the result of the research, namely:

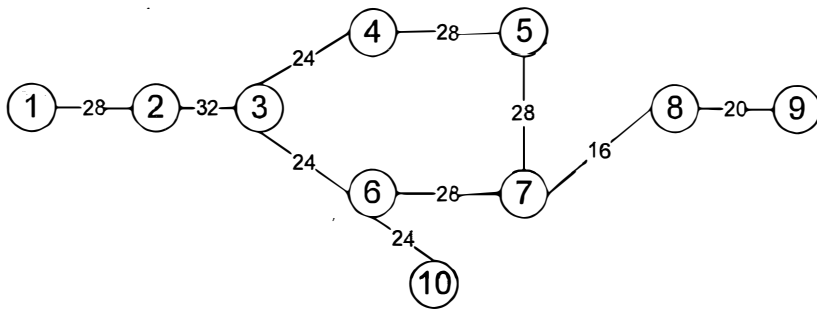
- CO graph
Each CO graph used in the research was produced by a clustering method of matrix permutation (Xiao, Zhang, & Jia, 2001). Use of different clustering methods can produce a different CO graph. The CO graph is the main input data used in the experiments.
- Spatial data used
Due to budget constraints, spatial data used in the experiments were generated artificially.
- The size of spatial data and clusters
The size of spatial data and clusters affect the cost of I/O in spatial join processing.
- The number of clusters
The number of clusters affects the AMO order and total amount of overlapping weight between two clusters.
- Time constraints
The research only took two semesters of study

3.2 Identification of Assumptions Underpinning the Study

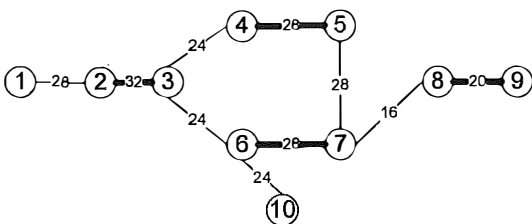
The study uses one of the concepts of graph theory, namely Maximum Weighted Matching. A maximum weighted matching problem is defined by a graph G consisting of a set V of vertices or nodes and a set of E of edges and the weights w . It is represented in the following equation, $G = (V, E, w)$. A matching on a graph is a set of edges such that no two of which meet at a common vertex. A maximum weighted matching is a matching for which the sum of the weights of the edges is maximal (Witwear, 2002).

Since the MBM method uses a greedy matching algorithm, which only has a performance ratio of 1:2 to maximum weighted matching, there is a room for

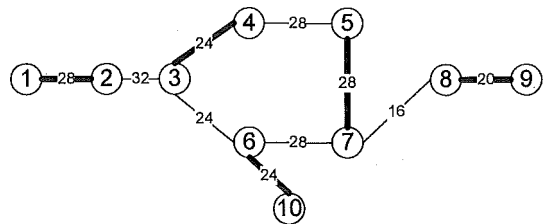
improvement if the maximum weighted matching algorithm is used instead of the greedy matching algorithm. Figure 11 shows a comparison of maximal matching result between greedy matching method and maximum weighted matching method. The greedy matching method finds 4 matchings with the total weight of 108 (32+28+28+20) whereas the maximum weighted matching method finds 5 matchings with the total weight of 124 (28+24+28+24+20). Thus, it can be said that using the maximum weighted matching algorithm will find a better AMO order with more total overlapping weight between clusters in a CO graph than using the greedy matching algorithm.



(a) An example of weighted graph



(b) greedy matching method



(c) maximum weighted matching method

Figure 11: Greedy matching vs. maximum weighted matching

The first polynomial algorithm for the weighted matching problem was introduced by Edmonds (1965). Its run time complexity is $O(n^2m)$, where n and m denote the number of vertices and edges in the graph respectively. Since then Edmonds' algorithm has been studied by a number of researchers. Gabow (1973) and Lawler (1976) have developed $O(n^3)$ implementation of Edmonds' algorithm. According to Vinkemeier and Hougardy (2005), the fastest implementations of Edmonds' algorithm are due to Cook and Rohe (1999) and Mehlhorn & Schäfer (2001) with a time complexity of $O(nm \log n)$.

Figure 12 summarises the algorithm of maximum weighted matching in pseudo-code (Witwear, 2002). See the Appendix B for more details about maximum weighted matching algorithm.

```

Initial Solution:
  let  $M$  be the empty matching
   $y_u = \max\{w_e/2 : e \in E\}$  for each vertex  $u \in G$ 
  label each vertex  $u \in G$  even
  for each vertex  $r \in G$  {
    if  $r$  is matched or  $y_r = 0$  continue
    let  $B_r$  be the only blossom of  $T$ 
    repeat {
      if an vertex  $u^+ \in T$  with  $y_u = 0$  exists {
        let  $P$  be the alternating path from  $u$  to  $r$ 
        replace  $M$  by  $m \oplus P$ 
      }
      else if an edge  $uv$  with  $u^+ \in T$  and  $\pi_{uv} = 0$ 
      exists {
        case  $v^* \notin T$ : grow step
        case  $v^+ \in T$ : shrink step
        case  $v^+ \notin T$ : augment step
      }
      else if there exists an odd blossom  $B^- \in T$ 
      with  $z_B = 0$ 
      expand step for  $B$ 
    }
    else {
      determine  $\delta$ 
      perform dual adjustment
    }
  }
  }
  }

```

Figure 12: Algorithm of maximum weighted matching

It has been identified by Xiao (2003) that the problem of finding maximum weighted matching in a non bipartite graph can guide the finding of AMO order in a CO graph. Thus, applying the maximum weighted matching algorithm to the proposed study may result in finding a better sequence of maximum overlapping clusters such that the sum of the edge weights of the edges or paths in the CO graph reaches the maximum. Hence, the I/O cost in spatial join processing can be minimised.

4 The Proposed Method

4.1 Maximal Weight Matching (MWM) Algorithm

The study proposes a new method called Maximal Weight Matching (MWM). This new MWM method is a cluster-scheduling method. It is similar to MST and MBM methods in the sense that their main objective is to find a better algorithm that produces a better sequence of clusters to guide the scheduling of clusters processing in a spatial join processing.

The MWM algorithm can be described in the following pseudo-code:

Algorithm: $mwmAMO(G)$

Input: $G = (V, E, w)$; a CO graph with $V = \{V_1, V_2, V_3, \dots, V_n\}$

Output: $V_{i1}, V_{i2}, \dots, V_{in}$; AMO order of G

Begin

1. Find a maximal match M of G using maximum weighted matching algorithm;
2. If no matching was found, return;
3. Add all matching into AMO;
4. Coarsen G by collapsing matching nodes of M to produce a coarser graph G' ;
5. $mwmAMO(G')$;
6. return;

End

The MWM method uses a recursive algorithm (see step 5). It can be separated into three main parts. In the first part, a maximal match M of G is produced using a maximum weighted matching algorithm (see algorithm details in Appendix B). Edges in M are taken as the initial AMO order.

In the second part, 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 all in the form of either $v = \{v_i, v_j\}$, where v_i and $v_j \in V$ are matched in M , i.e., $(v_i, v_j) \in M$, or $v = \{v_i\}$, where $v_i \in V$ is not matched in M , i.e., $(v_i) \notin M$. That is:

$$V' = \{ \{v_i, v_j\} \mid v_i, v_j \in V \text{ and } (v_i, v_j) \in M \} \cup \{ \{v_i\} \mid v_i \in V \text{ and } v_i \notin M \}$$

The node v of form $\{v_i, v_j\}$ in V' is called a multinode in Xiao (2003).

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. In other word, after collapsing the matching nodes, the edge with the highest weight is taken, among all edges connecting nodes in between the multinodes, to connect the new multinodes. In particular,

- For each pair of multinodes $v' \in V'$ and $v'' \in V'$, $v' \neq v''$, if $v' = \{v'_i, v'_j\}$, $v'' = \{v''_i, v''_j\}$, then $(v', v'') \in E'$ if and only if $w'(v', v'') \neq 0$, where $w'(v', v'') = \max \{w(u_1, u_2) \mid u_1 \in v', u_2 \in v'', (u_1, u_2) \in E\}$ (I)
- For each unmatched node $v = \{v_i\} \in V'$, if there exists a multinode $v' = \{v'_i, v'_j\}$ such that $(v_i, v'_i) \in E$ or $(v_i, v'_j) \in E$, then $(v, v') \in E'$ and $w'(v, v') = \max \{w(v_i, u) \mid u \in v', (v_i, u) \in E\}$ (II)

It is important to note that any pair of unmatched nodes is not connected in both G and G' .

After the graph G' is built, the maximum weighted matching algorithm is applied to G' again to produce a maximal match M' . The next level of coarser graph $G'' = (V'', E'', w'')$ can be built by the following procedure:

1. For each pair of matched nodes $v', v'' \in M'$,

- (1) If both v' and v'' are multinodes, i.e., each contains two nodes, say $v' = \{v'_i, v'_j\}$ and $v'' = \{v''_i, v''_j\}$, then choose a pair of nodes u' and u'' such that $u' \in v'$, $u'' \in v''$ and $w(u', u'') = \max \{w(u_1, u_2) \mid u_1 \in v', u_2 \in v'', (u_1, u_2) \in E\} \neq 0$,
 - i) add (u', u'') to AMO, and
 - ii) collapse the matched nodes v' and v'' by creating a new multinode $\{v', v''\} = \{v'_i, v'_j, v''_i, v''_j\}$.
- (2) If v' contains two nodes and v'' contains only one node, say $v' = \{v'_i, v'_j\}$ and $v'' = \{v''_k\}$, then choose $u \in v'$ such that $w(u, v''_k) = \max \{w(v'_i, v''_k), w(v'_j, v''_k)\}$,
 - i) add (u, v''_k) to AMO, and
 - ii) collapse nodes v' and v'' by creating a new multinode $\{v', v''\} = \{v'_i, v'_j, v''_k\}$.

(3) If v' contains one node only and v'' contains two nodes, similar to the case of (2).

(4) The V'' is created for the graph G'' which containing all multinodes created from the above steps.

2. For E'' and w'' , there are some remaining processes. These are:

- (1) define E'' and w'' similarly as those in (I) and (II), and
- (2) check whether the edges in E'' can potentially violate the rule of AMO order, that is, no cluster in an AMO order can have an edge degree more than 2, or no cluster in an AMO order can be linked to more than two other clusters. For example, in Figure 13 the cluster V3 is connected to more than two other clusters, which are cluster V1, V2, and V5, or it has an edge degree 3. Thus, it is not a valid AMO order. If there is an edge in E'' can potentially violate the rule, the edge is removed from the graph and go to step (1) again otherwise E'' and w'' are finalised for the graph G'' .

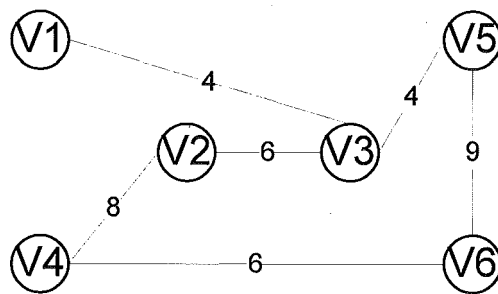


Figure 13: Example of invalid AMO order

The above matching and collapsing process continues until no further matching can be found. Finally, the algorithm produces the AMO order according to the AMO output from the above procedure.

4.2 The Complexity of MWM Algorithm

This section analyses the complexity of the proposed MWM algorithm detailed in section 4.1. For a given CO graph with n nodes, line 1 requires at most $O(n^3)$ time as it implements the Edmond's algorithm shown in Lawler, Lenstra, Kan, and Shmoys (1985). Line 3 needs $O(n^2)$ time as it scans at most once for each node to find its matching node. Lines 4 has the complexity of $O(n^2)$ because, for each matched node, it needs no more than once scanning to combine to its matched one to form a ultimode of the next level coarser graph. So the total complexity of lines 1-4 is $O(n^3)$. Line 5 completes the recursive execution of the algorithm.

Let $g(n)$ be the complexity function of the algorithm. First, consider the best case of the execution (that is 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, the recurrence formula for the complexity function will be

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

where α is a constant. As the collapsing reduces half the number of available (multi)nodes, either $n \leq 1$ 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

$$f(n) = \alpha \cdot n^3 + f\left(\frac{n}{2}\right) = \alpha \cdot n^3 + \alpha \cdot \left(\frac{n}{2}\right)^3 + f\left(\frac{n}{4}\right) = \dots = \alpha \cdot \left(n^3 + \left(\frac{n}{2}\right)^3 + \left(\frac{n}{4}\right)^3 + \dots + 1 \right) + f(1)$$

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

$$f(n) = \alpha n^3 \cdot \left(1 + \frac{1}{2^3} + \frac{1}{2^{2 \cdot 3}} + \frac{1}{2^{3 \cdot 3}} + \dots + \frac{1}{2^{3(k-1)}} \right) + f(1)$$

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

$$f(n) = O(n^3) \quad (III)$$

If n is not a power of 2, there must exist k such that $2k < n \leq 2(k+1)$. Therefore, we have $\frac{8}{7}\alpha.n^3 \leq f(n) \leq \frac{8}{7}.8\alpha.n^3$, which still leads to formula (III).

Secondly, consider the case where some unmatched (multi)nodes produced in step 1 of the MWM algorithm. Denote $F(n)$ as the complexity of the algorithm in this case. Without loss of generality, assume that the average number of matching pairs is $n/4$ ($\approx (1 + 2 + \dots + n/2) / (n/2)$). After collapsing, the next level of coarser graph will have $3n/4$ multinodes. In this case, we can get a recurrent function as

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

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 $n \cdot \left(\frac{3}{4}\right)^k = 1$, or $n = \left\lfloor \left(\frac{4}{3}\right)^k \right\rfloor$ for some k . According

to the recurrent relation of $F(n)$, we can derive

$$\begin{aligned} F(n) &= \alpha.n^3 + F\left(\frac{3}{4}n\right) = \alpha.n^3 + \alpha.\left(\frac{3}{4}n\right)^3 + F\left(\frac{3^2}{4^2}n\right) = \dots \\ &= \alpha.n^3 \left(1 + \frac{3^3}{4^3} + \frac{3^6}{4^6} + \frac{3^9}{4^9} + \dots\right) = \frac{64}{37}.\alpha.n^3 \end{aligned}$$

Or

$$F(n) \leq O(n^3) \quad (IV)$$

Following the discussion above, it can be shown that that formula (IV) holds for any n (Xiao, 2003). As the complexity of the algorithm is always greater-than-or-equal-to $f(n)$, and less-than-or-equal-to $F(n)$, i.e., $O(n^3) = f(n) \leq g(n) \leq F(n) \leq O(n^3)$, then the

complexity of $g(n) = O(n^3)$. The first research question will be quantified by the above means.

4.3 Comparison with Other Methods

To show how the proposed method works, the example of a CO graph with ten clusters in Figure 14 is used. As per research question 2, the example will also be used to compare to other methods, namely MST and MBM.

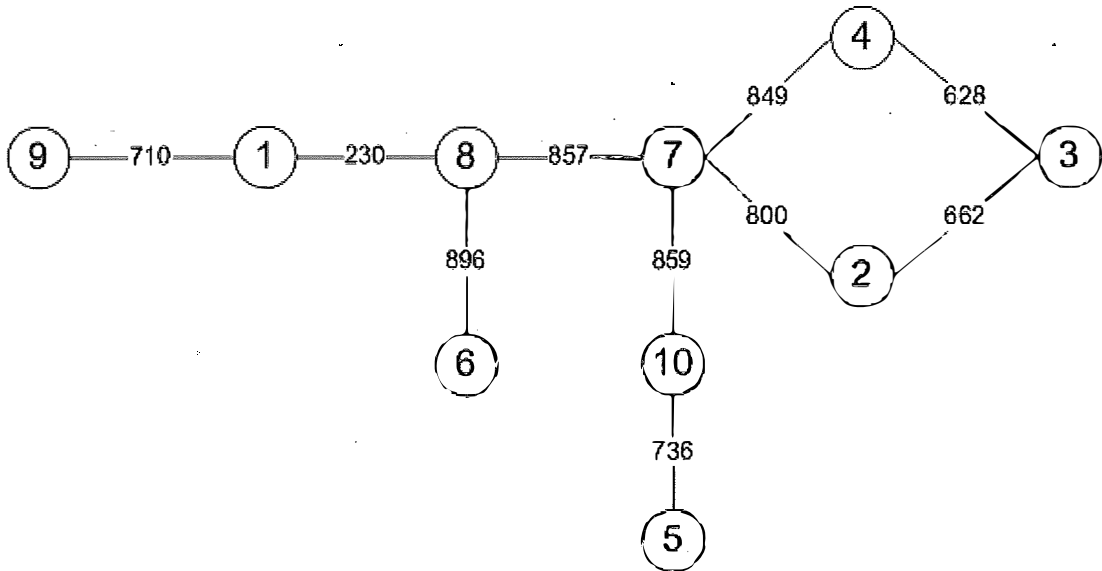
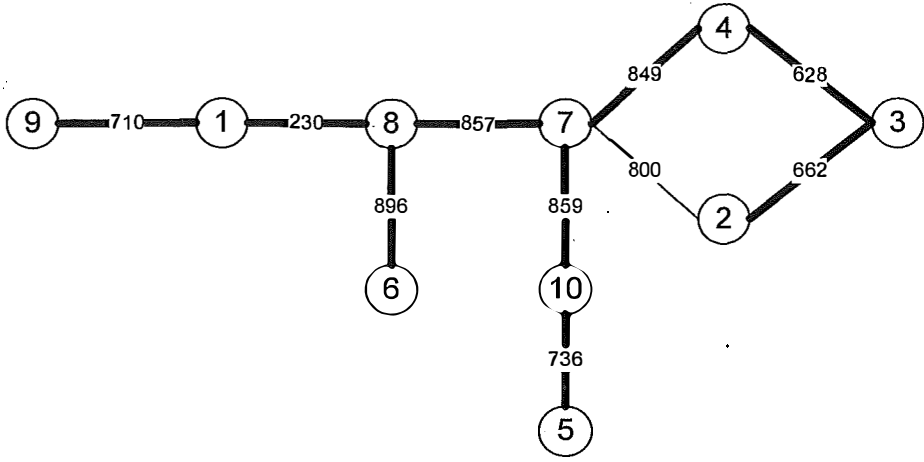
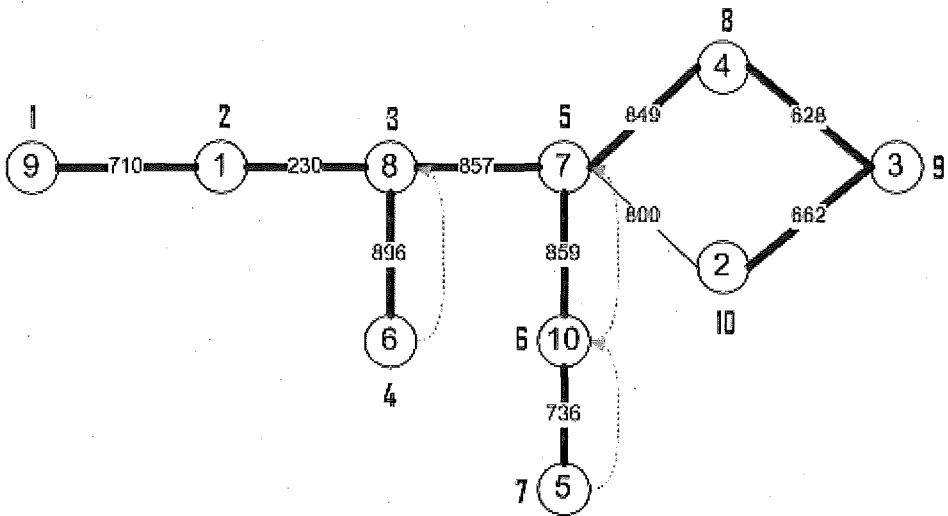


Figure 14: A CO graph with ten clusters

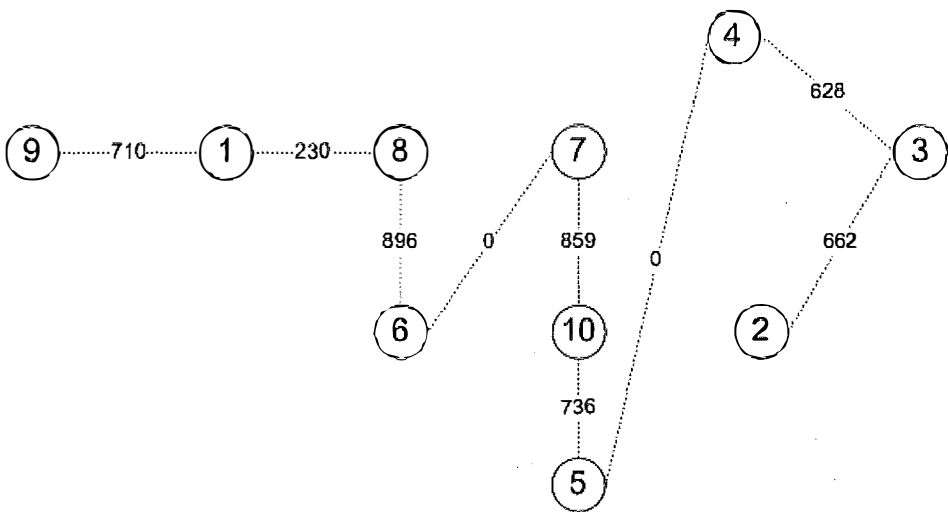
By applying the MST algorithm to the CO graph, the first step produces a maximum weight spanning tree T of the graph as shown in Figure 15(a). Suppose that the second step begins at cluster 9 in Figure 15(a). The result of the depth-first search of T is shown in Figure 15(b), where the numbers nearby the clusters are the traversal order numbers. In the third step, an AMO order is produced as shown in shown in Figure 15(c), which is $9 \rightarrow 1 \rightarrow 8 \rightarrow 6 \rightarrow 7 \rightarrow 10 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 2$, with the total overlapping weight of 4721.



(a) Maximal weighted spanning tree T



(b) A dept-first search of T



(c) An AMO order produced by MST

Figure 15: Execution of MST algorithm

By applying the MBM algorithm to the CO graph in Figure 14, an AMO order was produced as shown in Figure 16(e), which is $1 \rightarrow 9 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 10 \rightarrow 7 \rightarrow 8 \rightarrow 6$, with the total overlapping weight of 5348. The following shows the execution steps of the MBM algorithm (corresponding to Figure 16):

(a) The first step of the MBM algorithm uses the greedy matching algorithm to find the maximal match M . There are four pairs of matched nodes found:

- cluster 6 matches cluster 8 with edge weight 896
- cluster 7 matches cluster 10 with edge weight 859
- cluster 1 matches cluster 9 with edge weight 710
- cluster 3 matches cluster 2 with edge weight 662

Initial overlapping weight: 3127

Initial AMO order: (6, 8), (7, 10), (1, 9), (3, 2).

(b) The greedy matching algorithm is performed at the first level coarser graph. There are two pairs of matched nodes found:

- cluster 7 matches cluster 8 with edge weight 857
- cluster 4 matches cluster 3 with edge weight 628

Overlapping weight: 4612

AMO order: (6, 8), (7, 10), (1, 9), (3, 2), (7, 8), (4, 3).

(c) The greedy matching algorithm is performed at the second level coarser graph. There is a pair of matched nodes found:

- cluster 5 matches cluster 10 with edge weight 736

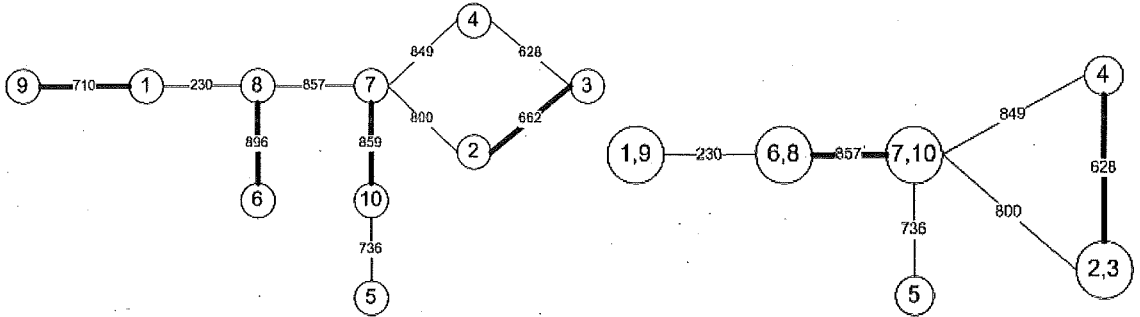
Overlapping weight: 5348

AMO order: (6, 8), (7, 10), (1, 9), (3, 2), (7, 8), (4, 3), (5, 10).

It is important to point out that the edges (7, 4) and (7, 2) with edge weights of 849 and 800 respectively, have been removed from the graph because they can potentially violate the rule of AMO order. Cluster 7 has been connected to cluster 10 in (a) and cluster 8 in (b), thus all edges that are incident to cluster 7 have to be removed to prevent producing an invalid AMO order. If the edge (7, 4) is not removed, the next matching will find cluster 7 matches to cluster 4, instead of cluster 10 matches cluster 5, because the weight of edge (7, 4) is 849 and it is heavier than the weight of edge (10, 5) which is 736 only. Thus, this will produce an invalid AMO order since the cluster 7 will have an edge degree 3.

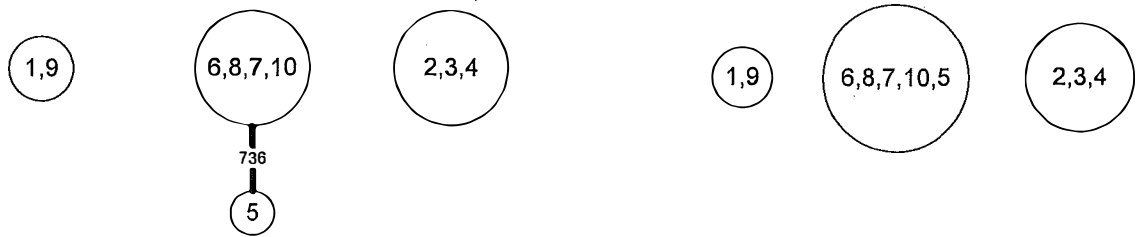
The same thing also happens to edge (1, 8).

- (d) The greedy matching algorithm is performed at the third level coarser graph. There is no pair of matched nodes found. The recursive algorithm part stops here.
- (e) The final compiled AMO order is $1 \rightarrow 9 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 10 \rightarrow 7 \rightarrow 8 \rightarrow 6$ with the total overlapping weight of 5348.



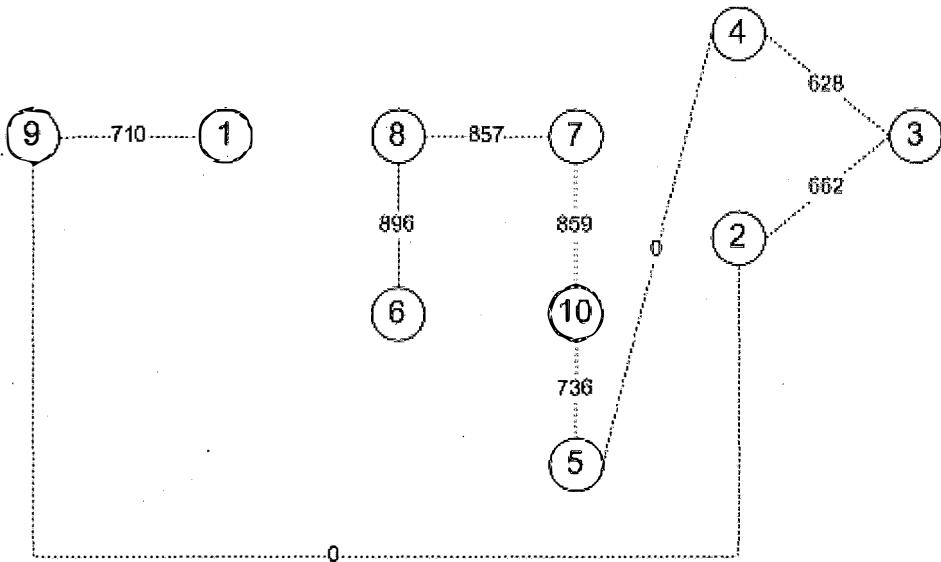
(a) First pairs of matched nodes

(b) A coarser graph and its pairs of matched nodes



(c) Next level coarser graph and its pair of matched nodes

(d) No matched nodes found



(e) An AMO order produced by MBM

Figure 16: Execution of MBM algorithm

By applying the MWM algorithm to the CO graph in Figure 14, an AMO order was produced as shown in Figure 17(e), which is $2 \rightarrow 3 \rightarrow 4 \rightarrow 7 \rightarrow 10 \rightarrow 5 \rightarrow 6 \rightarrow 8 \rightarrow 1 \rightarrow 9$, with the total overlapping weight of 5570. The following shows the execution steps of the MWM algorithm (corresponding to Figure 17):

- (a) The first step of the MWM algorithm uses the maximum weighted matching algorithm to find the maximal match M . There are five pairs of matched nodes found:

- cluster 1 matches cluster 9 with edge weight 710
- cluster 2 matches cluster 3 with edge weight 662
- cluster 4 matches cluster 7 with edge weight 849
- cluster 5 matches cluster 10 with edge weight 736
- cluster 6 matches cluster 8 with edge weight 896

Initial overlapping weight: 3853

Initial AMO order: (1, 9), (2, 3), (4, 7), (5, 10), (6, 8).

- (b) The maximum weighted matching algorithm is performed at the first level coarser graph. There are two pairs of matched nodes found:

- cluster 7 matches cluster 10 with edge weight 859
- cluster 1 matches cluster 8 with edge weight 230

Overlapping weight: 4942

AMO order: (1, 9), (2, 3), (4, 7), (5, 10), (6, 8), (7, 10), (1, 8).

- (c) The maximum weighted matching algorithm is performed at the second level coarser graph. There is a pair of matched nodes found:

- cluster 4 matches cluster 3 with edge weight 628

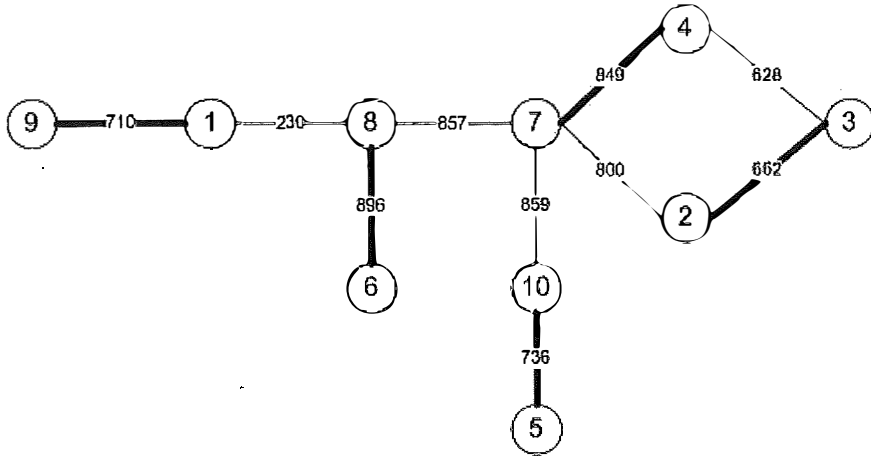
Overlapping weight: 5570

AMO order: (1, 9), (2, 3), (4, 7), (5, 10), (6, 8), (7, 10), (1, 8), (4, 3).

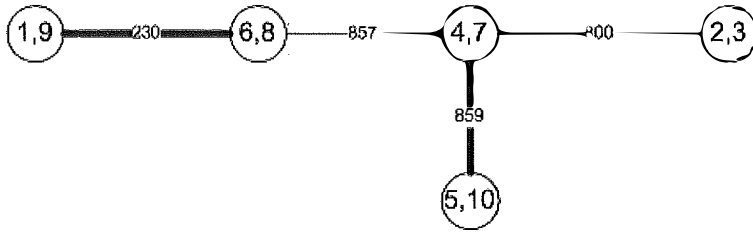
It is important to point out that the edges (8, 7) and (7, 2) with edge weights of 857 and 800 respectively, have been removed from the graph because they can violate the rule of AMO order. Cluster 7 has been connected to cluster 4 in (a) and cluster 10 in (b), thus all edges that are incident to cluster 7 have to be removed to prevent producing an invalid AMO order. As the edge (7, 2) is removed, the edge (4, 3) with the edge weight of 628 will appear to link the multinodes.

- (d) The maximum weighted matching algorithm is performed at the third level coarser graph. There is no pair of matched nodes found. The recursive algorithm part stops here.

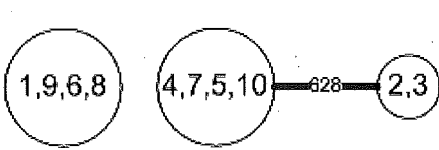
(e) The final compiled AMO order is $2 \rightarrow 3 \rightarrow 4 \rightarrow 7 \rightarrow 10 \rightarrow 5 \rightarrow 6 \rightarrow 8 \rightarrow 1 \rightarrow 9$ with the total overlapping weight of 5570.



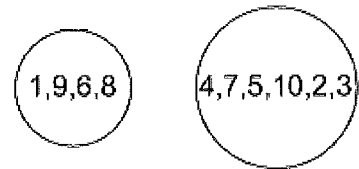
(a) First pairs of matched nodes



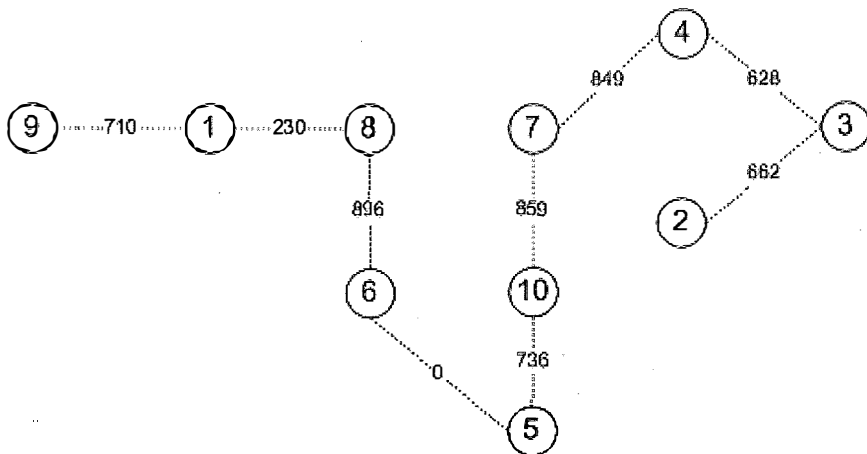
(b) A coarser graph and its pairs of matched nodes



(c) Next level coarser graph and its pair of matched nodes



(d) No matched nodes found



(e) An AMO order produced by MWM

Figure 17: Execution of MWM algorithm

To summarise this example, the total overlapping weight of:

- (a) MST method is 4721;
- (b) MBM method is 5348; and
- (c) MWM method is 5570.

Thus, from this comparison the initial findings are:

- (a) MBM produces 13.28% more overlapping weight than MST;
- (b) MWM produces 17.98% more overlapping weight than MST; and
- (c) MWM produces 4.15% more overlapping weight than MBM.

Clearly, MWM provides the best method of those compared.

5 Material and Methods

5.1 Design and Procedure of the Study

Figure 18 shows the overview of phases of the study. It started with literature review phase focusing on spatial join processing and different approaches to minimise the I/O cost involved in the process. Also, the graph theory of maximum weighted matching and its algorithm were studied both in bipartite and non bipartite graphs. The next phase is data collection/data generation. In this phase, spatial data were generated artificially. Then, these spatial datasets were clustered into CO graphs for further analysis by using a program that implements matrix-based algorithm of Xiao, Zhang, & Jia (2001).

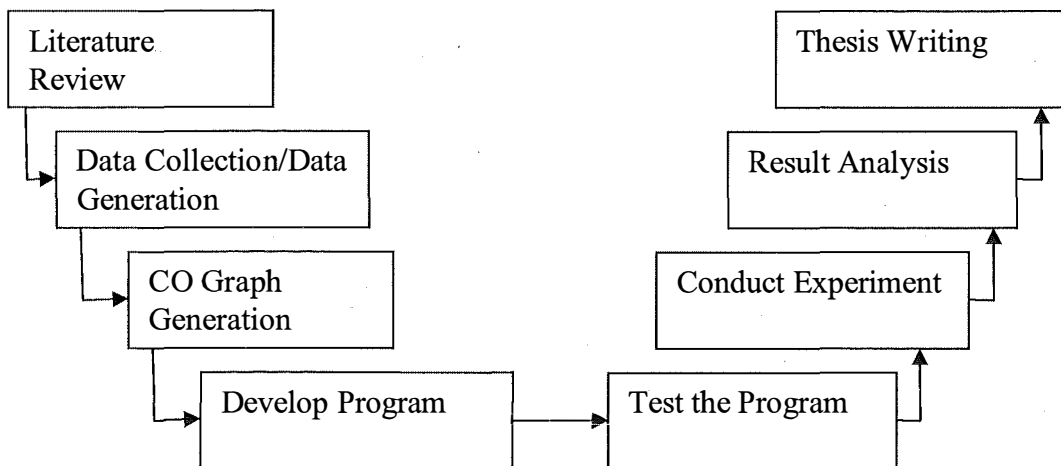


Figure 18: Procedure of the study

Next, a program was developed and tested during the second semester of the study. It was written in Java and implements the proposed algorithm and other methods, namely MST and MBM. It accepts the CO graph as an input and produces an output of AMO order for each method. The resulting AMO order can be used to guide the sequencing of the spatial join processing. After successful testing of the program, experiments were conducted and results were collected and analysed. Finally, the result of the experiments were interpreted and reported in this document.

5.2 Description of Instruments Employed

The research was conducted using an experiment research method. The experiment employed the use of a personal computer with JDK 1.5 installed and a Java IDE, Eclipse. Both of JDK 1.5 and Eclipse are open source tools that are free to use under the GPL license. These tools were used to develop a program for the experiments. The program implements all the algorithms used in the study, such as MST, MBM, and the proposed MWM methods. Some main functionalities of the program are:

- random CO graphs generation;
- manual creation of CO graphs;
- storing of a CO graph in an XML file; and
- producing an AMO order and the total overlapping weight for each method in a text file.

5.3 Data Analysis

Experiments were conducted after successful testing of the program. The experiments used the random generated CO graphs from ten clusters to a hundred clusters. For each cluster number, the experiments were conducted ten times with a different CO graph and the average result was taken. The collected results from each experiment are the AMO order and the total overlapping weight for each method. From the collected results, the performance of each method was compared by using the following three formulas:

$$1. \text{ MWM over MST} = \frac{\text{MWM} - \text{MST}}{\text{MST}} \times 100\%$$

$$2. \text{ MWM over MBM} = \frac{\text{MWM} - \text{MBM}}{\text{MBM}} \times 100\%$$

$$3. \text{ MBM over MST} = \frac{\text{MBM} - \text{MST}}{\text{MST}} \times 100\%$$

The first one is used to calculate how much percentage that the proposed MWM method outperforms the MST method in term of the total overlapping weight produced by each method. The second one is used to compare the proposed MWM method with the MBM. The last one is used to compare the MBM with the MST method.

Since all of the objects need to be fetched into the memory for the refinement step, for simplicity, the I/O cost can be referred as the total size of objects that are fetched into the memory for processing. Thus, the amount of total overlapping weight found in each method is the amount of I/O cost that can be saved if the clusters are processed in the sequence as the AMO order produced by each method because there is no need to fetch these overlapping objects when processing the next cluster as they are already in the memory.

6 Results and Findings

Before conducting experiments, there are two steps to be done, namely data collection and data clustering. The spatial data used in the experiments were artificially generated and then clustering of these data was performed using the clustering method of matrix-based algorithm of Xiao, Zhang, & Jia (2001). The result of the clustering method is CO graphs. These CO graphs were used as the main input for the experiments. From each of the CO graph, the experiment was conducted to find an AMO order and its total overlapping weight of each method, namely MST, MBM, and the proposed MWM.

The experiments were conducted to compare the proposed MWM method with MST and MBM methods. The experiments are to show how much the I/O cost can be saved by using MWM method and how well it can save the I/O cost comparing to MST and MBM methods.

Table 8 shows the experiment results with ten clusters CO graphs. There were ten experiments conducted with a different number of edges connecting the clusters. For example, for ten edges, the total overlapping weight produced by MST, MBM, and MWM method are 4721, 5348, and 5570 respectively. Thus, MBM outperforms MST by 13.28% and the proposed MWM method outperforms MST and MBM by 17.98% and 4.15% respectively. The average results for ten clusters CO graphs showed that the proposed MWM method can potentially produce 5.59% and 16.91% more total overlapping weight comparing to MBM and MST respectively.

Table 8: Results of experiment with 10 clusters

| Cluster Number | Edge | MST | MBM | MWM | MWM over MBM | MWM over MST | MBM over MST |
|----------------|------|--------|--------|--------|--------------|--------------|--------------|
| 10 | 10 | 4721 | 5348 | 5570 | 4.15% | 17.98% | 13.28% |
| 10 | 15 | 3869 | 4027 | 4302 | 6.83% | 11.19% | 4.08% |
| 10 | 20 | 5596 | 6405 | 6648 | 3.79% | 18.80% | 14.46% |
| 10 | 22 | 5416 | 6122 | 6536 | 6.76% | 20.68% | 13.04% |
| 10 | 25 | 5624 | 6782 | 7144 | 5.34% | 27.03% | 20.59% |
| 10 | 30 | 5774 | 6015 | 6532 | 8.60% | 13.13% | 4.17% |
| 10 | 33 | 5575 | 6120 | 6300 | 2.94% | 13.00% | 9.78% |
| 10 | 35 | 6542 | 7442 | 7882 | 5.91% | 20.48% | 13.76% |
| 10 | 40 | 6686 | 7035 | 7548 | 7.29% | 12.89% | 5.22% |
| 10 | 45 | 6944 | 7583 | 7910 | 4.31% | 13.91% | 9.20% |
| Average: | | 5674.7 | 6287.9 | 6637.2 | 5.59% | 16.91% | 10.76% |

Figure 19 shows the results of the Table 8 in a bar chart. It can be seen that the proposed method performs all the time better than the other two methods. For example, when the edge number is 35, the total overlapping weight found by MST and MBM are 6542 and 7442 respectively while it is 7882 by using the proposed method.

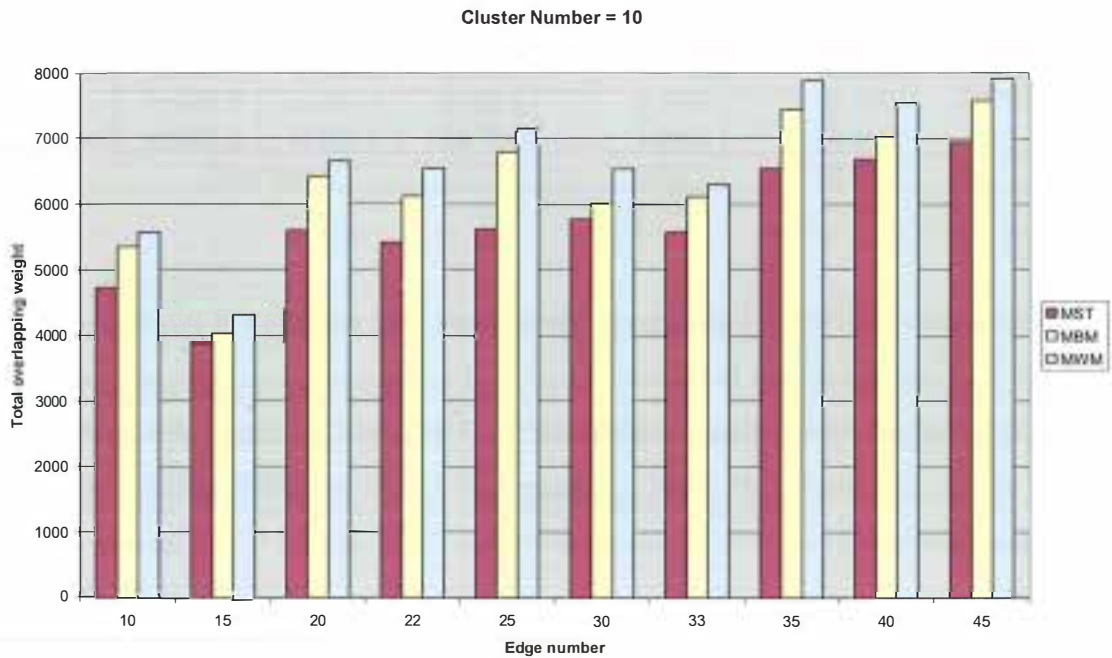


Figure 19: Comparison of total overlapping weight

Table 9 shows the summary result of the experiments. For each cluster number, there were ten experiments conducted and the average results are shown in the table. For example, for ten clusters, the average of total overlapping weight produced by MST, MBM, and MWM method are 5674.7, 6287.9, and 6637.2 respectively and the percentage average of performance comparison for each method is also shown in the table (see Appendix C for details of each experiment).

Table 9: Summary of experiment results

| Cluster number | MST | MBM | MWM | MWM over MBM | MWM over MST | MBM over MST |
|----------------|---------|---------|---------|--------------|--------------|--------------|
| 10 | 5674.7 | 6287.9 | 6637.2 | 5.59% | 16.91% | 10.76% |
| 20 | 9509.8 | 10401.4 | 11038.9 | 6.12% | 16.18% | 9.49% |
| 30 | 14068.9 | 15103.9 | 15708.7 | 4.05% | 12.11% | 7.77% |
| 40 | 17592.4 | 18980.3 | 19819 | 4.34% | 13.28% | 8.60% |
| 50 | 21539.5 | 23530 | 24431.6 | 3.75% | 14.09% | 10.00% |
| 60 | 31011.8 | 33060.1 | 34424.5 | 4.11% | 11.05% | 6.67% |
| 70 | 38011 | 41211.5 | 42448.2 | 3.03% | 12.28% | 8.98% |
| 80 | 38738.5 | 41552.1 | 42870.6 | 3.20% | 10.69% | 7.28% |
| 90 | 41586.7 | 44306.5 | 45569.5 | 2.78% | 9.78% | 6.82% |
| 100 | 43001.9 | 47588.9 | 48517.4 | 1.94% | 12.95% | 10.80% |
| Average | | | | 3.89% | 12.93% | 8.72% |

Figure 20 shows the average total amount of I/O cost can be saved by each method in a line chart. As expected, the proposed method performs all the time better than the other two methods. On average, there are 12.93% saving comparing with the MST and 3.89% saving comparing with the MBM. For example, when the cluster number is 50, the average saving of I/O cost by MST and MBM are 21539.5 and 23530 respectively while it is 24431.6 by using the proposed method.

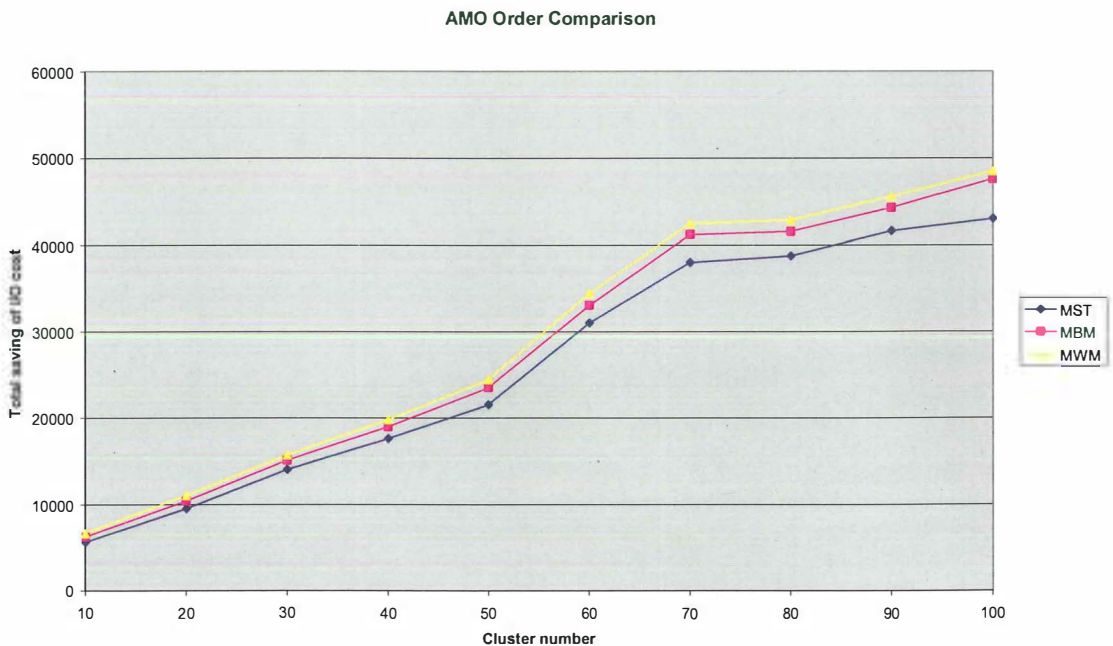


Figure 20: Comparison of total saving of I/O cost

7 Further Study

The main limitation of the study is the use of generated artificially spatial data in the experiments due to budget constraints. Another limitation is the time frame of the research. The research only took two semesters of study. Finally, the research does not cover the clustering method.

To overcome these limitations, future research can be done which may include:

- (a) The use of real world spatial data in the experiments;
- (b) Covering the concept of clustering technique and perform the clustering to the real spatial data; and
- (c) The implementation of a new cluster-scheduling method that uses other types of algorithms such as Ant Colony Optimisation algorithm or Genetic algorithms to find a better sequence to schedule the cluster loading process in the spatial join processing.

8 Conclusion

The cost of the spatial join processing is far greater than relational join processing. The reasons are the size of spatial objects is generally very large and it requires expensive computations of spatial join predicates. Generally, the spatial join processing is processed in two steps: filtering and refinement steps. Since the filtering step operates on approximations of the actual spatial data, the result is not final but a set of candidates that are likely to satisfy the spatial join predicate. In the refinement step, full geometry descriptions of these candidates are fetched into memory and then the spatial join operation is performed on the actual geometry. Fetching these spatial objects from disks into the memory requires a lot of I/O cost. The objective of this research is to minimise the I/O cost at the refinement step so that the spatial join query can be performed efficiently.

A cluster-scheduling technique is one the most successful techniques to minimise the I/O cost of a spatial join processing. The main point of this technique is to partition the candidate sets, which are a result from the filter step, into several clusters and then to schedule the processing of the clusters in an order such that the two consecutive clusters in the sequence have maximal number of overlapping objects. Thus, there is no need to fetch these overlapping objects again when processing the next cluster because they are already in the memory. Consequently, the I/O cost can be minimised.

The proposed MWM method is based on the cluster-scheduling technique. It performs better than other cluster-scheduling methods, namely MST and MBM, in terms of producing a better AMO order to guide the cluster scheduling. The experiments have been conducted and the results have shown that, in terms of I/O cost, MWM outperforms MST by 13% and MBM by 4%.

The proposed MWM method is a new method that can further minimise the I/O cost of spatial join processing. Hence, it results in faster and more efficient processing of a spatial join query in spatial databases. As the demand of using spatial databases is increasing, this proposed method is certainly significant.

References

- Abel, D. J., Gaede, V., Power, R. A., & Zhou, X. (1999). Caching Strategies for Spatial Joins. *GeoInformatica*, 3(1), 33 - 59
- Arge, L., Procopiuc, O., Ramaswamy, S., Suel, T., & Vitter, J. S. (1998). Scalable Sweeping-Based Spatial Join. *Proceedings of the 24th International Conference on Very Large Data Bases*, 570-581.
- Beckmann, N., Begel, H. P., Schneider, R., & Seeger, B. (1990). The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 322-331.
- Black, P. E. (2005). Dictionary of Algorithms and Data Structures Retrieved 7 June, 2006, from <http://www.nist.gov/dads/>
- Brinkhoff, T., Kriegel, H. P., & Seeger, B. (1993). Efficient Processing of Spatial Joins Using R-Trees. In *Proceeding of the 1993 ACM SIGMOD Int. Conf. on Management of Data*, 237-246.
- Chan, C. Y., & Ooi, B. C. (1997). Efficient Scheduling of Page Access in Index-Based Join Processing. *IEEE transactions on knowledge and data engineering*, 9(6), 1005-1011.
- Cook, W. J., Cunningham, W. H., Pulleyblank, W. R., & Schrijver, A. (1998). *Combinatorial Optimization*. New York: John Wiley & Sons, Inc.
- Cook, W. J., & Rohe, A. (1999). Computing Minimum-Weight Perfect Matchings. *INFORMS journal on computing*, 11(2), 138.
- Dittrich, J. P., & Seeger, B. (2000). Data Redundancy and Duplicate Detection in Spatial Join Processing. *Proceedings of the 16th International Conference on Data Engineering*, 535.
- Doratha, E. D., & Hougardy, S. (2003). A Simple Approximation Algorithm for the Weighted Matching Problem. *Information Processing Letters*, 85(4), 211-213.
- Edmonds, J. (1965). Path, Tree, and Flower. *Journal of Math*, 17, 449-467.
- Fotouhi, F., & Pramanik, S. (1989). Optimal Secondary Storage Access Sequence for Performing Relational Join. *IEEE Transactions on Knowledge and Data Engineering* 1(3), 318-328.
- Gabow, H. N. (1973). *Implementation of Algorithms for Maximum Matching on Nonbipartite Graphs*. Unpublished PhD thesis, Stanford University.
- Gütting, R. H. (1994). An introduction to spatial database systems. *The VLDB Journal* 3(4), 357-399.
- Guttman, A. (1984). R-trees: a dynamic index structure for spatial searching. *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, 47-57.

- Hoel, E. G., & Samet, H. (1992). A Qualitative Comparison Study of Data Structures for Large Linear Segment Databases. *In Proc. ACM-SIGMOD International Conference on Management of Data*, 205-214.
- Hoel, E. G., & Samet, H. (1995). Benchmarking Spatial Join Operations with Spatial Output. *Proc. of the 21st Intl. Conf. on Very Large Databases*, 606-618.
- Howe, D. (2006). FOLDOC: Free On-Line Dictionary of Computing. Retrieved 7 June 2006, from <http://foldoc.org/>
- Huang, Y. W., Jing, N., & Rundensteiner, E. A. (1997). Spatial Joins Using R-trees: Breadth-First Traversal with Global Optimizations. *Proceedings of the 23rd VLDB Conference*, 396-405.
- Huang, Y. W., Jones, M., & Rundensteiner, E. A. (1998). Symbolic Intersect Detection: A Method for Improving Spatial Intersect Joins. *GeoInformatica*, 2(2), 149-174.
- Koudas, N., & Sevcik, K. C. (1997). Size Separation Spatial Join. *Proceedings of the 1997 ACM SIGMOD international conference on Management of data* 324-335.
- Lawler, E. L. (1976). *Combinatorial Optimization: Networks and Matroids*. New York: Holt, Rinehart, and Winston.
- Lawler, E. L., Lenstra, J. K., Kan, A. H. G. R., & Shmoys, D. B. (1985). *The Travelling Salesman Problem*. Essex: John Wiley and Sons.
- Leutenegger, S. T., Edgington, J. M., & Lopez, M. A. (1997). STR: A Simple and Efficient Algorithm for R-Tree Packing. *Proc. Int'l Conf. Data Eng*, 497-506.
- Lim, A., Pheng, K. L., & Chong, O. W. (2001). Page access scheduling in join processing. *Data & knowledge engineering*, 37(3), 267-284.
- Lo, M. L., & Ravishankar, C. V. (1994). Spatial Joins Using Seeded Trees. *Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, 209-220.
- Lo, M. L., & Ravishankar, C. V. (1996). Spatial Hash-Joins. *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, 247-258.
- Mamoulis, N., & Papadias, D. (2003). Slot Index Spatial Join. *IEEE Transactions on Knowledge and Data Engineering*, 15(1), 211-231.
- Mehlhorn, K., & Schäfer, G. (2001). *Implementation of $O(nm \log n)$ Weighted Matchings in General Graphs: The Power of Data Structures*. Paper presented at the Proceedings of the 4th International Workshop on Algorithm Engineering, Springer-Verlag.
- Neyer, G., & Widmayer, P. (1997). Singularities Make Spatial Join Scheduling Hard. *Lecture Notes in Computer Science*, 1350 293-302.
- Omiecinski, E. R. (1989). Heuristics for Join Processing Using Nonclustered Indexes. *IEEE Trans. Softw. Eng.*, 15(1), 18-25.

- Orenstein, J. (1990). A comparison of spatial query processing techniques for native and parameter spaces. *In ACM SIGMOD Int. Conf. on Management of Data*, 326-336.
- Papadopoulos, A., Rigaux, P., & Scholl, M. (1999). A Performance Evaluation of Spatial Join Processing Strategies. *Proc. Int'l Symp. Large Spatial Databases*, 286-307.
- Patel, J., & DeWitt, D. (1996). Partition based spatial-merge join. *In Proc. of ACM SIGMOD*, 259-270.
- Poirier, G. (2002). Spatially Augmented Greedy Heuristic for Efficient I/O in Spatial-join Processing of Non-uniform-sized Spatial objects. Retrieved 1 April, 2006, from <http://www.cgl.uwaterloo.ca/~gpoirier/cs741/SpatialJoin.pdf>
- Sellis, T., Roussopoulos, N., & Faloutsos, C. (1987). The R⁺-Tree: A Dynamic Index for Multi-Dimensional Objects. *In Proc. IEEE International Conf. on Very Large Databases*, 507-518.
- Shekhar, S., Lu, C. T., Chawla, S., & Ravada, S. (2002). Efficient Join-Index-Based Spatial-Join Processing: A Clustering Approach. *IEEE transactions on knowledge and data engineering*, 14(6), 1400.
- Shekhar, S., Lu, C. T., Ravada, S., & Chawla, S. (1998). Optimizing Join Indexed Based Join Processing: A Graph Partitioning Approach. *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*, 302-310.
- Stonebraker, M., Frew, J., & Dozier, J. (1993). The SEQUOIA 2000 Project. *In Proceedings of the Third International Symposium on Large Spatial Databases*, 397-412.
- Valduriez, P. (1987). Join Indices. *ACM Trans. Database Syst.*, 12(2), 218-246
- Vassilakopoulos, M., & Corral, A. (2005). Spatio-Temporal Indexing Techniques. *Encyclopedia of Database Technologies and Applications*, 652-657.
- Vinkemeier, D. E. D., & Hougardy, S. (2005). A linear-time approximation algorithm for weighted matchings in graphs. *ACM Transactions on Algorithms*, 1(1), 107-122.
- Witwear, C. (2002). Maximum Weighted Matching Algorithm. Retrieved May 2, 2006, from <http://www.tbi.univie.ac.at/Leere/Slides/814236/ws02/xtina0.pdf>
- Xiao, J. (2003). An Efficient Algorithm for Scheduling Spatial Join Operations in Spatial Database Systems. *Software Engineering and Applications*, 397.
- Xiao, J., Zhang, Y., & Jia, X. (2001). Clustering Non-uniform-sized Spatial Objects to Reduce I/O Cost for Spatial-join Processing. *The Computer Journal*, 44(5), 384-397.
- Xiao, J., Zhang, Y., Jia, X., & Zhou, X. (2000). A Schedule of Join Operations to Reduce I/O Cost in Spatial Database Systems. *Data and Knowledge Engineering*, 35, 299-307.

Appendix A – Definitions of Terms

| Term | Description | Source |
|-----------------|---|--|
| Algorithm | A detailed sequence of problem-solving procedure, especially an established, recursive computational procedure for solving a problem in a finite number of steps. | (Howe, 2006) |
| AMO order | Approximate Maximum Overlapping order. A sequence, which is close to MO order, represents clusters loading process to minimise the I/O cost of fetching objects. | (Xiao, Zhang, Jia, & Zhou, 2000) |
| Bipartite graph | A graph is bipartite if its nodes can be partitioned into two sets V_1 and V_2 so that every edge joins a node in V_1 to a node in V_2 . | (Cook, Cunningham, Pulleyblank, & Schrijver, 1998) |
| CO graph | Cluster Overlapping graph. A weighted graph represents the overlapping relationships between data clusters. | (Xiao, Zhang, Jia, & Zhou, 2000) |
| GIS | Geographic Information System. A computer system for capturing, storing, checking, integrating, manipulating, analysing, and displaying data related to some space. | (Howe, 2006) |
| GPL | General Public Licence. The licence is intended to guarantee the freedom to share and change free software. The GPL allows users to distribute the software and its source code freely. | (Howe, 2006) |
| Hyper graph | A hyper graph G can be defined as a pair (V, E) , where V is a set of vertices, and E is a set of hyper edges between the vertices. Each hyper edge is a set of vertices: $E = \{\{u, v, \dots\} \in 2V\}$. Hyper edges are undirected. | (Black, 2005) |
| I/O | Input/Output. Communication between a computer and its users, its storage devices, other computers (via a network) or the outside world. Important aspects of I/O are throughput, latency, and whether the communications is synchronous or asynchronous (using some kind of buffer). | (Howe, 2006) |
| MBM | Match Based Method. MBM uses a greedy matching algorithm to find an AMO order in a CO graph. | (See page 28) |
| MBR | Minimum Bounding Rectangle. A common used method to approximate spatial objects. | (Xiao, Zhang, Jia, & Zhou, 2000) |
| MO order | Maximum Overlapping order. An MO order in a CO graph is a permutation of nodes in the graph such that the total size of overlapping objects between adjacent nodes reaches the maximum. | (Xiao, Zhang, Jia, & Zhou, 2000) |
| MST | Maximum Spanning Tree. MST schedules clusters in three steps: find a maximum weight spanning tree, conduct a depth-first-search of the tree, and construct an AMO order. | (Xiao, Zhang, Jia, & Zhou, 2000) |
| MWM | Maximal Weight Matching. MWM uses a | (See page 33) |

| | | |
|----------------|---|---------------------------------------|
| | maximum weighted matching algorithm to find an AMO order in a CO graph. | |
| NP | Non-deterministic Polynomial-time. The complexity class of decision problems for which answers can be checked by an algorithm whose run time is polynomial in the size of the input. | (Black, 2005) |
| NP-complete | A problem is NP-complete if it is both NP (verifiable in non-deterministic polynomial time) and NP-hard (any NP-problem can be translated into this problem). | (Black, 2005) |
| NP-hard | A problem is NP-hard if solving it in polynomial time would make it possible to solve all problems in class NP in polynomial time. | (Black, 2005) |
| OPAS-FB | Optimal Page Access Sequence with Fix Buffer. It concerns with finding the optimal page access sequence that minimises the number of page re-accesses given a fixed buffer size. | (Lim, Pheng, & Chong, 2001) |
| Path graph | A path graph $G = (V, E)$ with n nodes is a graph in which nodes in V can be listed as a sequence $v_1, v_2, \dots, v_{n-1}, v_n$ such that $(v_1, v_2), (v_2, v_3), \dots, (v_{n-1}, v_n)$ are the only edges of E . | (Xiao, 2003) |
| PCG | Page Connectivity Graph. A bipartite graph represents a join index between two relations at the page level. | (Shekhar, Lu, Chawla, & Ravada, 2002) |
| Quadtree | A four-way tree where each node corresponds to a sub-quadrant of the quadrant of each parent node (the root corresponds to the whole space). | (Vassilakopoulos & Corral, 2005) |
| R-tree | A balance multiway tree for secondary storage, where each node is related to a MBR, the minimum rectangle that bounds the data elements contained in the node. The MBR of the root bounds all the data stored in the tree. | (Vassilakopoulos & Corral, 2005) |
| Weighted graph | A graph having a weight or number associated with each edge. | (Black, 2005) |

Appendix B – $O(n^3)$ Weighted Matching Algorithm

The following Weighted Matching algorithm with the running time complexity of $O(n^3)$ is implemented in the program developed for the experiments. It is adopted from Lawler (1976). The steps are:

Step 0 (Start)

The graph $G = (N, A)$ is given, with a weight w_{ij} for each arc (i, j) . Let $W = \frac{1}{2} \max \{w_{ij}\}$. Set $u_i = W$, $\gamma_i = \pi_i = \tau_i = +\infty$ and $b(i) = i$ for each node $i \in N$. For each node pair i, j set $C(i, j) = \Phi$. Set $X = \Phi$. There are no blossoms and no nodes are labelled.

Step 1 (Labelling)

(1.0) Apply the label “S: Φ ” to each exposed node.

(1.1) If there is no node i with an unscanned S-label or an unscanned T-label with $\pi_i = 0$ go to Step 4. Otherwise, find such a node i . If the label is an S-label, go to Step 1.2; if it is a T-label, go to Step 1.3.

(1.2) Scan the S-label on node i by carrying out the following procedure for each arc $(i, j) \notin X$ incident to node i :

- If $b(i) = b(j)$, do nothing; otherwise continue.
- If node $b(j)$ has an S-label and $\bar{w}_{ij} = 0$, backtrace from the S-labels on nodes i and j . If different root nodes are reached, go to Step 2; if the same root node is reached, go to Step 3.
- If node $b(j)$ has an S-label and $\bar{w}_{ij} > 0$, then carry out the following procedure. Set

$$\gamma_{b(i)} = \min \left\{ \gamma_{b(i)}, \frac{1}{2} \bar{w}_{ij} \right\},$$

$$\gamma_{b(j)} = \min \left\{ \gamma_{b(j)}, \frac{1}{2} \bar{w}_{ij} \right\}.$$

Find $C(b(i), b(j)) = (p, q)$. If $\bar{w}_{ij} < \bar{w}_{pq}$, then set $C(b(i), b(j)) = (i, j)$.

- If node $b(j)$ has no S-label and $\bar{w}_{ij} < \pi_{b(j)}$, then apply the label “T: i, j ” to $b(j)$, replacing any existing T-label, and set $\pi_{b(j)} = \bar{w}_{ij}$.
- If node $b(j)$ has no S-label and $\bar{w}_{ij} < \tau_j$, then set $\tau_j = \bar{w}_{ij}$ and set $f(j) = i$.
- When the scanning of node i is complete, return to Step 1.1.

(1.3) Scan the T-label on node i (where $\pi_i = 0$) by carrying out the following procedure for the unique arc $(i, j) \in X$ incident to node i .

- If $b(i) = b(j)$, do nothing; otherwise continue.
- If node j has a T-label and $\pi_j = 0$, backtrack from the T-labels on nodes i and j . If different root nodes are reached, go to Step 2; if the same root node is reached, go to Step 3.
- Otherwise, give node j the label “S: i .” The S-labels on all nodes within the outermost blossom with base node j are now considered to be unscanned.
- Return to Step 1.1.

Step 2 (Augmentation)

An augmenting path has been found in Step 1.2, 1.3, or 4.2. Augment the matching X . Correct labels on nodes in the augmenting path, as described in the text. Expand blossoms with zero dual variables, resetting the blossom numbers. Remove labels from all base nodes. The remaining labels are set to the “scanned” state. Set $\gamma_i = \pi_i = \tau_i = +\infty$ for all i , and $C(i, j) = \Phi$, for all i, j . Go to Step 1.0.

Step 3 (Blossoming)

A blossom has been formed in Step 1.2, 1.3, or 4.2. Determine the membership and base node of the new blossom, as described in the text. Supply the missing labels for all nodes, except the base node, in the new blossom. Reset the blossom numbers. Set the z -variable 1.0 zero for the new blossom.

Let b be the base node of the new blossom, and I be the set of (old) base nodes contained in the blossom. Let J be the complementary set of base nodes. For each $j \in J$, find arc $C(i, j) = (p', q')$, for which $\bar{w}_{p'q'} = \min_{i \in I} \{\bar{w}_{pq} \mid C(i, j) = (p, q)\}$, and set $C(b, j) = (p', q')$. Then set $\gamma_b = \min_{j \in J} \{\bar{w}_{pq} \mid C(b, j) = (p, q)\}$.

Return to Step 1.2, 1.3, or 4.2, as appropriate.

Step 4 (Revision of Dual Solution)

(4.1) Let K_S denotes the set of S-blossoms and K_T denotes the set of T-blossoms, i.e., outermost blossoms whose base nodes b have T-labels with $\pi_b = 0$.

Find

$$\delta_1 = \min \{u_i\}$$

$$\delta_2 = \frac{1}{2} \min \{Z_k \mid k \in K_T\}$$

$$\delta_3 = \min \{\gamma_i \mid b(i) = i\}$$

$$\delta_4 = \min \{ \pi_i \mid \pi_i > 0 \}$$

$$\delta = \min \{ \delta_1, \delta_2, \delta_3, \delta_4 \}.$$

Set $u_i = u_i - \delta$, for each node i such that $b(i)$ has an S-label.

Set $u_i = u_i + \delta$, for each node i such that $b(i)$ has a T-label and $\pi_{b(i)} = 0$.

Set $\gamma_i = \gamma_i - 2\delta$, for each node i such that $b(i) = i$.

Set $\pi_i = \pi_i - \delta$, if $\pi_i > 0$.

Set $\tau_i = \tau_i - \delta$, for each node i such that $\pi_{b(i)} > 0$.

Set $Z_k = Z_k - 2\delta$, for each blossom $k \in K_T$.

Set $Z_k = Z_k + 2\delta$, for each blossom $k \in K_S$.

If $\delta = \delta_1$ halt; X is a maximum weight matching, and the values of u_i, Z_k yield an optimal solution.

If $\delta = \delta_2$, carry out the following procedure to expand each T-blossom k for which $Z_k = 0$. Determine the blossoms nested immediately within the T-blossom and reset $b(i)$ for all nodes within the blossom. Remove labels from all new base nodes within the blossom. For each new base node b , find $\tau_i = \min \{ \tau_j \mid b(j) = b \}$, and if $\tau_i < +\infty$, apply the (unscanned) label "T:t(i), i" to b and set $\pi_b = \tau_i$. Remaining labels on nodes within the blossom are in a "scanned" state.

(4.2) If $\gamma_b > 0$, for all base nodes b , go to Step 1.1. Otherwise, find a base node b for which $\gamma_b = 0$ and a base node b' such that $\bar{w}_{ij} = 0$ for $(i, j) = C(b, b')$. Backtrace from the S-labels on i and j . If different root nodes are reached, go to Step 2. If the same root node is reached, go to Step 3, later returning to Step 4.2.

Appendix C – Experimental Results

All results of the experiments are shown in the following tables. The MST, MBM, and MWM columns show the amount of total overlapping weight produced by each method respectively. To know how they are calculated, see the example in section 4.3. The cluster number is the number of nodes whereas the edge is the number of links connecting nodes in a CO graph.

Experiment with 10 clusters

| Cluster Number | Edge | MST | MBM | MWM | MWM over MBM | MWM over MST | MBM over MST |
|----------------|------|--------|--------|--------|--------------|--------------|--------------|
| 10 | 10 | 4721 | 5348 | 5570 | 4.15% | 17.98% | 13.28% |
| 10 | 15 | 3869 | 4027 | 4302 | 6.83% | 11.19% | 4.08% |
| 10 | 20 | 5596 | 6405 | 6648 | 3.79% | 18.80% | 14.46% |
| 10 | 22 | 5416 | 6122 | 6536 | 6.76% | 20.68% | 13.04% |
| 10 | 25 | 5624 | 6782 | 7144 | 5.34% | 27.03% | 20.59% |
| 10 | 30 | 5774 | 6015 | 6532 | 8.60% | 13.13% | 4.17% |
| 10 | 33 | 5575 | 6120 | 6300 | 2.94% | 13.00% | 9.78% |
| 10 | 35 | 6542 | 7442 | 7882 | 5.91% | 20.48% | 13.76% |
| 10 | 40 | 6686 | 7035 | 7548 | 7.29% | 12.89% | 5.22% |
| 10 | 45 | 6944 | 7583 | 7910 | 4.31% | 13.91% | 9.20% |
| Average: | | 5674.7 | 6287.9 | 6637.2 | 5.59% | 16.91% | 10.76% |

Experiment with 20 clusters

| Cluster Number | Edge | MST | MBM | MWM | MWM over MBM | MWM over MST | MBM over MST |
|----------------|------|--------|---------|---------|--------------|--------------|--------------|
| 20 | 20 | 7161 | 7635 | 7922 | 3.76% | 10.63% | 6.62% |
| 20 | 25 | 7649 | 8357 | 9029 | 8.04% | 18.04% | 9.26% |
| 20 | 30 | 7485 | 8091 | 8603 | 6.33% | 14.94% | 8.10% |
| 20 | 35 | 9258 | 10658 | 11556 | 8.43% | 24.82% | 15.12% |
| 20 | 40 | 9928 | 10898 | 11752 | 7.84% | 18.37% | 9.77% |
| 20 | 45 | 8426 | 9657 | 10125 | 4.85% | 20.16% | 14.61% |
| 20 | 50 | 11213 | 11851 | 12723 | 7.36% | 13.47% | 5.69% |
| 20 | 55 | 10996 | 12104 | 12431 | 2.70% | 13.05% | 10.08% |
| 20 | 60 | 12050 | 12840 | 13839 | 7.78% | 14.85% | 6.56% |
| 20 | 65 | 10932 | 11923 | 12409 | 4.08% | 13.51% | 9.07% |
| Average: | | 9509.8 | 10401.4 | 11038.9 | 6.12% | 16.18% | 9.49% |

Experiment with 30 clusters

| Cluster Number | Edge | MST | MBM | MWM | MWM over MBM | MWM over MST | MBM over MST |
|----------------|------|---------|---------|---------|--------------|--------------|--------------|
| 30 | 30 | 9831 | 11440 | 12052 | 5.35% | 22.59% | 16.37% |
| 30 | 35 | 11055 | 11435 | 11909 | 4.15% | 7.73% | 3.44% |
| 30 | 40 | 13412 | 14103 | 15114 | 7.17% | 12.69% | 5.15% |
| 30 | 45 | 12564 | 14363 | 14696 | 2.32% | 16.97% | 14.32% |
| 30 | 50 | 15226 | 16998 | 17480 | 2.84% | 14.80% | 11.64% |
| 30 | 55 | 15064 | 16672 | 17106 | 2.60% | 13.56% | 10.67% |
| 30 | 60 | 15050 | 15526 | 16132 | 3.90% | 7.19% | 3.16% |
| 30 | 65 | 14959 | 15654 | 16135 | 3.07% | 7.86% | 4.65% |
| 30 | 70 | 15647 | 16862 | 17307 | 2.64% | 10.61% | 7.77% |
| 30 | 75 | 17881 | 17986 | 19156 | 6.51% | 7.13% | 0.59% |
| Average: | | 14068.9 | 15103.9 | 15708.7 | 4.05% | 12.11% | 7.77% |

Experiment with 40 clusters

| Cluster Number | Edge | MST | MBM | MWM | MWM over MBM | MWM over MST | MBM over MST |
|----------------|------|---------|---------|-------|--------------|--------------|--------------|
| 40 | 40 | 11387 | 12258 | 12687 | 3.50% | 11.42% | 7.65% |
| 40 | 45 | 14889 | 17476 | 18155 | 3.89% | 21.94% | 17.38% |
| 40 | 50 | 13711 | 16213 | 16633 | 2.59% | 21.31% | 18.25% |
| 40 | 55 | 19629 | 20824 | 21564 | 3.55% | 9.86% | 6.09% |
| 40 | 60 | 18584 | 19526 | 20917 | 7.12% | 12.55% | 5.07% |
| 40 | 65 | 16093 | 17842 | 18324 | 2.70% | 13.86% | 10.87% |
| 40 | 70 | 17723 | 19376 | 20311 | 4.83% | 14.60% | 9.33% |
| 40 | 75 | 21993 | 22990 | 23898 | 3.95% | 8.66% | 4.53% |
| 40 | 80 | 19226 | 20172 | 21612 | 7.14% | 12.41% | 4.92% |
| 40 | 85 | 22689 | 23126 | 24089 | 4.16% | 6.17% | 1.93% |
| Average: | | 17592.4 | 18980.3 | 19819 | 4.34% | 13.28% | 8.60% |

Experiment with 50 clusters

| Cluster Number | Edge | MST | MBM | MWM | MWM over MBM | MWM over MST | MBM over MST |
|----------------|------|---------|-------|---------|--------------|--------------|--------------|
| 50 | 50 | 15702 | 17875 | 18213 | 1.89% | 15.99% | 13.84% |
| 50 | 55 | 17316 | 20682 | 21051 | 1.78% | 21.57% | 19.44% |
| 50 | 60 | 19896 | 22553 | 23087 | 2.37% | 16.04% | 13.35% |
| 50 | 65 | 20967 | 23233 | 23690 | 1.97% | 12.99% | 10.81% |
| 50 | 70 | 19587 | 22734 | 24435 | 7.48% | 24.75% | 16.07% |
| 50 | 75 | 26802 | 27600 | 28740 | 4.13% | 7.23% | 2.98% |
| 50 | 80 | 20960 | 22201 | 23362 | 5.23% | 11.46% | 5.92% |
| 50 | 85 | 27138 | 27554 | 29028 | 5.35% | 6.96% | 1.53% |
| 50 | 90 | 24184 | 27228 | 28078 | 3.12% | 16.10% | 12.59% |
| 50 | 95 | 22843 | 23640 | 24632 | 4.20% | 7.83% | 3.49% |
| Average: | | 21539.5 | 23530 | 24431.6 | 3.75% | 14.09% | 10.00% |

Experiment with 60 clusters

| Cluster Number | Edge | MST | MBM | MWM | MWM over MBM | MWM over MST | MBM over MST |
|----------------|------|---------|---------|---------|--------------|--------------|--------------|
| 60 | 80 | 27175 | 29042 | 30102 | 3.65% | 10.77% | 6.87% |
| 60 | 90 | 25163 | 27176 | 28248 | 3.94% | 12.26% | 8.00% |
| 60 | 100 | 29339 | 32213 | 32919 | 2.19% | 12.20% | 9.80% |
| 60 | 110 | 31862 | 33742 | 34748 | 2.98% | 9.06% | 5.90% |
| 60 | 120 | 33489 | 35198 | 36732 | 4.36% | 9.68% | 5.10% |
| 60 | 130 | 31143 | 32563 | 34637 | 6.37% | 11.22% | 4.56% |
| 60 | 140 | 33416 | 35918 | 37680 | 4.91% | 12.76% | 7.49% |
| 60 | 150 | 34569 | 36296 | 37651 | 3.73% | 8.92% | 5.00% |
| 60 | 160 | 31860 | 33851 | 34957 | 3.27% | 9.72% | 6.25% |
| 60 | 170 | 32102 | 34602 | 36571 | 5.69% | 13.92% | 7.79% |
| Average: | | 31011.8 | 33060.1 | 34424.5 | 4.11% | 11.05% | 6.67% |

Experiment with 70 clusters

| Cluster Number | Edge | MST | MBM | MWM | MWM over MBM | MWM over MST | MBM over MST |
|----------------|------|-------|---------|---------|--------------|--------------|--------------|
| 70 | 90 | 27918 | 31172 | 31833 | 2.12% | 14.02% | 11.66% |
| 70 | 110 | 31102 | 34180 | 35951 | 5.18% | 15.59% | 9.90% |
| 70 | 130 | 30647 | 38222 | 39199 | 2.56% | 27.90% | 24.72% |
| 70 | 150 | 37810 | 39273 | 40544 | 3.24% | 7.23% | 3.87% |
| 70 | 170 | 38332 | 41465 | 42544 | 2.60% | 10.99% | 8.17% |
| 70 | 190 | 39600 | 41894 | 43315 | 3.39% | 9.38% | 5.79% |
| 70 | 210 | 41818 | 42779 | 44318 | 3.60% | 5.98% | 2.30% |
| 70 | 230 | 42776 | 47413 | 48873 | 3.08% | 14.25% | 10.84% |
| 70 | 250 | 43770 | 47508 | 48336 | 1.74% | 10.43% | 8.54% |
| 70 | 270 | 46337 | 48209 | 49569 | 2.82% | 6.97% | 4.04% |
| Average: | | 38011 | 41211.5 | 42448.2 | 3.03% | 12.28% | 8.98% |

Experiment with 80 clusters

| Cluster Number | Edge | MST | MBM | MWM | MWM over MBM | MWM over MST | MBM over MST |
|----------------|------|---------|---------|---------|--------------|--------------|--------------|
| 80 | 100 | 32381 | 34742 | 35855 | 3.20% | 10.73% | 7.29% |
| 80 | 110 | 34576 | 38444 | 38979 | 1.39% | 12.73% | 11.19% |
| 80 | 120 | 35270 | 37198 | 39501 | 6.19% | 12.00% | 5.47% |
| 80 | 130 | 38812 | 40395 | 41028 | 1.57% | 5.71% | 4.08% |
| 80 | 140 | 39363 | 42472 | 43921 | 3.41% | 11.58% | 7.90% |
| 80 | 150 | 41086 | 42205 | 43818 | 3.82% | 6.65% | 2.72% |
| 80 | 160 | 39100 | 42795 | 44077 | 3.00% | 12.73% | 9.45% |
| 80 | 170 | 42684 | 46322 | 46915 | 1.28% | 9.91% | 8.52% |
| 80 | 180 | 41430 | 43491 | 45421 | 4.44% | 9.63% | 4.97% |
| 80 | 190 | 42683 | 47457 | 49191 | 3.65% | 15.25% | 11.18% |
| Average: | | 38738.5 | 41552.1 | 42870.6 | 3.20% | 10.69% | 7.28% |

Experiment with 90 clusters

| Cluster Number | Edge | MST | MBM | MWM | MWM over MBM | MWM over MST | MBM over MST |
|----------------|------|---------|---------|---------|--------------|--------------|--------------|
| 90 | 100 | 30025 | 34371 | 35046 | 1.96% | 16.72% | 14.47% |
| 90 | 110 | 37294 | 39753 | 40938 | 2.98% | 9.77% | 6.59% |
| 90 | 120 | 38608 | 39922 | 40662 | 1.85% | 5.32% | 3.40% |
| 90 | 130 | 41001 | 44286 | 45351 | 2.40% | 10.61% | 8.01% |
| 90 | 140 | 42037 | 46058 | 47057 | 2.17% | 11.94% | 9.57% |
| 90 | 150 | 42097 | 45622 | 46365 | 1.63% | 10.14% | 8.37% |
| 90 | 160 | 44784 | 46519 | 47708 | 2.56% | 6.53% | 3.87% |
| 90 | 170 | 45938 | 48311 | 49462 | 2.38% | 7.67% | 5.17% |
| 90 | 180 | 48377 | 50715 | 53686 | 5.86% | 10.97% | 4.83% |
| 90 | 190 | 45706 | 47508 | 49420 | 4.02% | 8.13% | 3.94% |
| Average: | | 41586.7 | 44306.5 | 45569.5 | 2.78% | 9.78% | 6.82% |

Experiment with 100 clusters

| Cluster Number | Edge | MST | MBM | MWM | MWM over MBM | MWM over MST | MBM over MST |
|----------------|------|---------|---------|---------|--------------|--------------|--------------|
| 100 | 100 | 36677 | 42920 | 43868 | 2.21% | 19.61% | 17.02% |
| 100 | 110 | 39200 | 44525 | 45195 | 1.50% | 15.29% | 13.58% |
| 100 | 120 | 41271 | 45618 | 46216 | 1.31% | 11.98% | 10.53% |
| 100 | 130 | 37322 | 40118 | 41008 | 2.22% | 9.88% | 7.49% |
| 100 | 140 | 38688 | 42653 | 43335 | 1.60% | 12.01% | 10.25% |
| 100 | 150 | 45528 | 51353 | 52124 | 1.50% | 14.49% | 12.79% |
| 100 | 160 | 48052 | 49188 | 50182 | 2.02% | 4.43% | 2.36% |
| 100 | 170 | 47270 | 53043 | 54145 | 2.08% | 14.54% | 12.21% |
| 100 | 180 | 48678 | 54882 | 56040 | 2.11% | 15.12% | 12.74% |
| 100 | 190 | 47333 | 51589 | 53061 | 2.85% | 12.10% | 8.99% |
| Average: | | 43001.9 | 47588.9 | 48517.4 | 1.94% | 12.95% | 10.80% |

Appendix D – CD Contents

The attached CD contains the following:

1. Folder 'AMO'

This folder contains everything for a user who is interested to run the program. It has two components:

- AMO: an executable JAR file which is the program itself.
- Folder 'project' which is the folder for storing the generated xml file.

2. User manual

The user manual covers the installation and program guides.

3. Folder 'Experiment results'

This folder contains two main parts:

- All of the graphs used in the experiment and their results for all the methods.
- 'AMO Comparison' is an MS Excel file containing all the collected results, their summary in tables, and charts to show the performance comparison of each method.