

Exploiting Spatial Indexes for Semijoin-Based Join Processing in Distributed Spatial Databases

Kian-Lee Tan, *Member, IEEE Computer Society*,
Beng Chin Ooi, *Member, IEEE Computer Society*, and David. J. Abel

Abstract—In a distributed spatial database system, a user may issue a query that relates two spatial relations not stored at the same site. Because of the sheer volume and complexity of spatial data, spatial joins between two spatial relations at different sites are expensive in terms of computation and transmission cost. In this paper, we address the problems of processing spatial joins in a distributed environment. We propose a semijoin-like operator, called the *spatial semijoin*, to prune away objects that will not contribute to the join result. This operator also reduces both the transmission and local processing costs for a later join operation. However, the cost of the elimination process must be taken into account, and we consider approaches to minimize these overheads. We also studied and compared two families of distributed join algorithms that are based on the spatial semijoin operator. The first is based on multidimensional approximations obtained from an index such as the R-tree, and the second is based on single-dimensional approximations obtained from object mapping. We conducted experiments on real data sets and report the results in this paper.

Index Terms—Spatial indexes, R-tree, locational keys, distributed spatial database systems, spatial semijoin, query processing.



1 INTRODUCTION

QUERIES in spatial databases frequently involve relationships between two spatial entities. These relationships include *containment*, *intersection*, *adjacency*, and *proximity*. For example, the query “which schools are adjacent to areas zoned for industrial purposes?” requires an adjacency relationship, while the query “which police stations are within 1 kilometer from a major road?” involves a proximity relationship. To answer these queries, spatial joins are used to materialize the relationships between the two spatial entities. Like the relational join operation, spatial joins are expensive and have received much research attention recently [5], [9], [8], [11], [12], [14], [15], [17], [18], [27], [32].

While spatial database research to date has largely focused on single-site databases, some prospective applications now call for distributed spatial databases. A representative example is in government agencies where the sharing of core data sets across agencies has been shown to provide high savings [33]. Legislative and organizational requirements make it very difficult to establish a single unified database; rather, data sharing must be approached as a problem in distributed access to many autonomous databases.

To realize the full potential of a distributed spatial database system, many issues have to be addressed. Most of these issues are similar to those that arise in designing heterogeneous database systems [30]. These include the integration of existing schemas and data, the processing and optimization of distributed queries, and transaction processing issues. However, the issue of processing distributed spatial query has been largely ignored.

In a distributed spatial database system, a user may issue a query that joins two spatial relations stored at different sites. The sheer volume and complexity of spatial data lead to expensive spatial join processing across sites in terms of computation and transmission cost. In this paper, we focus on the design of distributed spatial join algorithms. Unlike conventional distributed databases where the transmission cost is dominant [26], both the transmission cost and the processing cost may be comparable for distributed spatial databases. Thus, it is no longer appropriate to design algorithms that minimize transmission cost alone. Instead, we design and study distributed spatial join algorithms that are based on the concept of *spatial semijoins*. A spatial semijoin eliminates objects before transmission to reduce both transmission and local processing costs. This elimination is performed as a spatial selection on one database using *approximations* to the spatial descriptions of the other. Inherently, the elimination process itself introduces additional costs and, hence, the choice of approximation is critical to the performance of the distributed algorithms. By varying the approximations to the spatial descriptions, we can study the tradeoffs between transmission cost and processing cost.

In our earlier paper [4], we proposed a distributed semijoin-based strategy that employs single-dimensional approximations obtained from object mapping as a possible

- K.-L. Tan and B.C. Ooi are with the Department of Computer Science, National University of Singapore, Lower Kent Rd., Singapore 119260. E-mail: ooibc@comp.nus.edu.sg.
- D.J. Abel is with CSIRO Mathematical and Information Sciences, GPO Box 664, Canberra, ACT 2601, Australia.

Manuscript received 4 Dec. 1997; revised 12 July 1999; accepted 16 Aug. 1999.

For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 106006.

join optimization strategy in our heterogeneous spatial database system [3]. The technique allows us to exploit the sort-merge algorithm. In this paper, we built on and extended this initial work. Besides reporting more experimental studies on the algorithms, we also propose a new family of join strategies that employ the spatial semijoin operator. The new algorithms are based on multidimensional approximations obtained from an index. We study R-tree index and use the bounding boxes stored in an existing R-tree as approximations. We conducted extensive experiments on our semijoin-based algorithms on real data sets. The results show that the methods are effective in reducing total processing cost in distributed join processing. We also report on a comparative study on the two families of algorithms.

The remainder of this paper is organized as follows: In the next section, we shall look at the issues involved in distributed spatial join processing. We also review uniprocessor spatial join algorithms. Section 3 introduces the concept of spatial semijoin, and we present a framework for designing distributed spatial join algorithms. In Section 4, we describe the two families of join strategies studied. In Section 5, we present and analyze the results from experiments with a representative database, and finally, we summarize our conclusions in Section 6.

2 SPATIAL JOIN PROCESSING

In this section, we review uniprocessor join algorithms. In particular, we pay more attention to join algorithms that employ R-trees [13] and locational key techniques [10], as our distributed spatial join strategies are based on them.

2.1 Uniprocessor Join Processing: A Brief Review

Several spatial join processing algorithms have been studied. Experimental and analytical results have shown that algorithms that employ indexing or ordering techniques are much more efficient than simple nested-loops joins [9], [11], [12], [17]. Before looking at some of these schemes, it is worth pointing out that several spatial join algorithms have recently been proposed for nonindexed relations—*Partition-Based Merge Join* [27], *Spatial Hash Join* [18], *Size Separation Spatial Join* [15], and *Scalable Sweeping-Based Spatial Join* [5].

2.1.1 Joins Based on Multidimensional Indexing Techniques

Spatial joins can be performed efficiently by exploiting existing indexes. Rotem [28] applied the concept of *join indexes* [34] on two frequently joined spatial relations indexed by grid files [22]. Lu and Han also proposed a similar mechanism using *distance* metrics to facilitate fast access to spatial window queries [20]. The *approximate* join indexes [19] speeds up join processing by precomputing pairs of index pages that contain matching objects. In [32], a join algorithm for relations indexed with *filter trees* is proposed.

Several spatial join algorithms based on R-tree-like indexes have also been studied [12], [9], [17]. The R-tree [13] and its variants (R⁺-tree [31], R*-tree [7]) have been widely used to speed up access for multidimensional

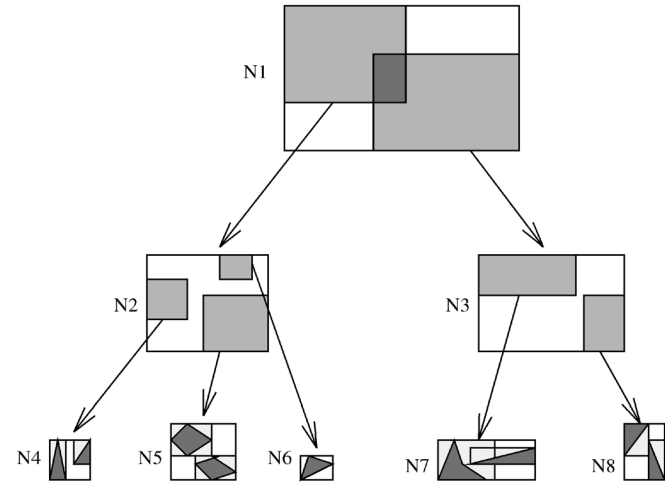


Fig. 1. An R-tree index.

spatial objects. Like the B⁺-tree, the R-tree is height-balanced. Each internal R-tree node contains entries of the form $(mbr, childptr)$, where *mbr* represents the minimum bounding rectangle (MBR) enclosing all the objects described by the child node pointed to by *childptr*. A leaf node contains entries of the form (mbr, oid) , where *mbr* is the MBR of the object that *oid* points to. Fig. 1 illustrates an index structure of the R-tree.

Spatial join algorithms based on R-trees require that both relations be indexed. If a relation is not indexed, then one will be created temporally for join processing [17]. The basic idea is to traverse both spatial indexes simultaneously, and entries of internal nodes are checked to see if they overlap. If the condition is true, both the subtrees are recursively traversed. Results are produced when the leaf nodes are reached. In [12], a breadth-first-search algorithm is adopted in traversing the tree, while a depth-first-search algorithm is employed in [9], [17].

2.1.2 Joins Based on Linearized Single-Dimensional Object Mapping

Linearizing spatial objects has the advantage that sort-merge join methods can be exploited. Sort-merge join methods require, in the best case, that both data sets be scanned at most once. Various techniques on ordering multidimensional objects using single-dimensional values have been proposed [10], [23] and one of the most popular ordering techniques is that based on bit interleaving proposed by Morton [21].

In [23], a space is recursively divided into four equal-sized quadrants as in the quad-tree [29], forming a hierarchy of quadrants. These quadrants are then linearized based on their z-ordering (see Fig. 2), and objects can be accessed quickly using one-dimensional indexes (such as the B⁺-tree) on their z-values. The join between two relations is performed using a merge-like operation on the z-values of the objects [24], [25].

A similar approach is adopted in [10]. However, for each quadrant, a unique key of base 5 is attached. Fig. 2 illustrates an example of key assignment. As can be seen from the figure, when these keys are traced, it is the z-order enumeration of grid cells as in [23]. An object that is fully

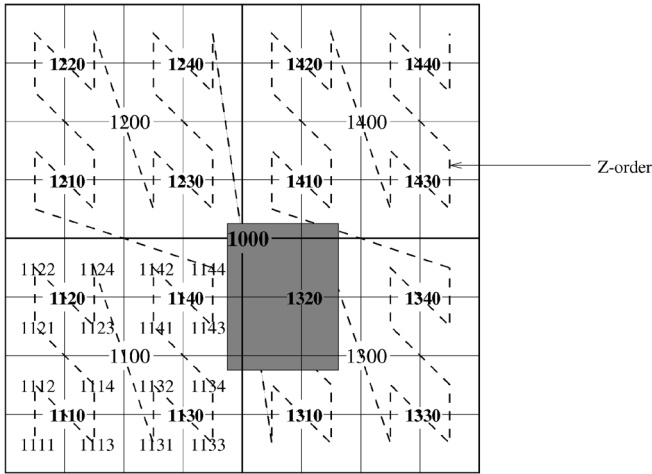


Fig. 2. Assignment of locational keys.

contained by a subspace is assigned its key. To improve the approximation of objects by quadrants, an object is assigned up to k locational keys (In [10], $k = 4$). For example, using $k = 4$, the rectangular object in Fig. 2 will be assigned the keys: 1,100, 1,233, 1,300, 1,410. The spatial objects are held sorted in ascending sequence by the locational key value, with each member of the sorted list consisting of the object identifier, its MBR, and the assigned locational key. A B^+ -tree is used to provide direct access to the spatial objects based on the locational keys [1].

Consider two spatial relations R and S that are to be joined. Fig. 3a shows the objects in R and S , and Fig. 3b shows the locational key values of the objects. In this example, we have allowed each object to have a maximum of four keys. Joining the two relations can be performed by a merge-like operation along the two sorted lists of locational keys. We note the following. First, the join is a nonequi join. For example, the locational key value 1,240 of $S4$ matches two locational key values of $R1$ (namely, 1,241 and 1,242). Second, the results may contain duplicates. This is because each object has multiple keys, and different pairs of keys of the same objects may intersect. For example, there are two resulting pairs between $R1$ and $S4$. Finally, the results may contain *false drops*, i.e., results obtained from

matching locational keys but the actual objects do not match. Looking at Fig. 3a, we notice that there are only two intersections, but we obtain four. The pairs $(R1, S4)$ and $(R2, S2)$ are false drops. This arises because the locational keys are but approximations of the actual objects.

2.2 Issues in Distributed Spatial Join Processing

While much research has been done in distributed query processing (see [26], [37]), the unique characteristics of spatial data (large volume of large data items and complex operations) pose interesting challenges.

It traditionally has been assumed that transmission of data dominates distributed query processing performance and, as a result, many algorithms have been designed to minimize transmission cost [26]. This assumption is valid in the old days when network speeds were slow and wide area networks were assumed. While the transmission cost of exporting a spatial data set from one site to another can be high, this assumption may no longer be valid in distributed spatial databases. This is because of the high local processing cost for spatial operations. The local processing cost cannot be ignored since

- many spatial databases of real-world interest are very large, with sizes ranging from tens of thousands to millions of objects;
- spatial descriptions of objects are typically extensive, ranging from a few hundred bytes in Land Information System applications to megabytes in natural resource applications [2]; and
- many basic spatial operations, such as testing the intersection of two polygons, are expensive.

As a broad indication of the relative costs of operations, transmission of a land parcel spatial description (with an average size of 48 bytes) over an Ethernet Local Area Network requires 0.5 milliseconds. On the other hand, retrieving a polygon from a database of 50,000 polygons using an intersection qualification costs in the region of 2 milliseconds on a SUN SPARC-10 machine. The challenge then is to develop new distributed spatial join strategies that take into account the transmission cost as well as the local processing cost.

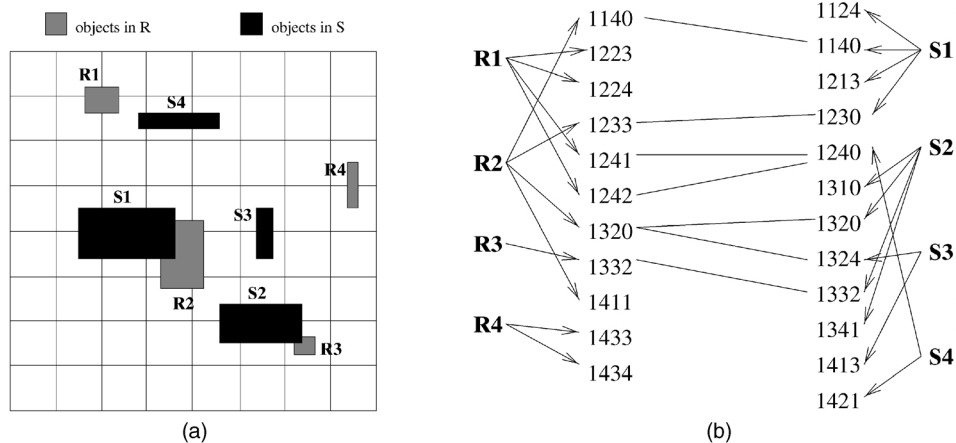


Fig. 3. Local join processing using locational keys. (a) Sample objects from two relations to be joined. (b) Sort-merge join processing.

In this paper, unless otherwise stated, R and S represent two spatial relations. R resides at site R_{site} and S at site S_{site} . A spatial join between R and S is on the spatial attributes $R.A$ and $S.B$. The result of the spatial join is to be produced at site S_{site} .

3 A FRAMEWORK FOR SEMIJOIN-BASED SPATIAL JOIN

A straightforward approach to perform a spatial join between R and S is to transmit the whole of R from R_{site} to S_{site} . The spatial join can then be performed at S_{site} using an existing uniprocessor join algorithm. This method, though simple, incurs high transmission cost and high local processing cost.

In this section, we present an alternative approach that is based on the concept of semijoin. We will first look at the semijoin operator for spatial databases, called the *spatial semijoin*, and then present a framework for designing semijoin-based algorithms.

3.1 Spatial Semijoin

In conventional distributed databases, the semijoin operator [6] has been proposed to reduce transmission costs. The θ -semijoin of a relation R with another relation S on the join condition $R.A \theta S.B$ is the relation $R' \subseteq R$ such that all the records in R' satisfy the join condition, where θ is a scalar comparison operator (i.e., $<$, $>$, $<=$, $>=$, $=$). (In [6], θ is set to " $=$ ".) The join of R and S , that are located at sites R_{site} and S_{site} , respectively, can be performed using a semijoin in three steps. First, S is projected on the joining attribute at S_{site} , and the resultant set of distinct values, say S' , is transmitted to R_{site} . Next, the semijoin of R and S' is performed at R_{site} to give R' , which is sent to S_{site} . Finally, the join of R' and S is performed at S_{site} to produce the join result. Since R' has fewer records than R , transmitting R' is cheaper. Moreover, there is some saving in local processing costs for evaluation of the final join at site S_{site} . Clearly, there are additional local processing and transmission operations to be performed and a design objective is to ensure that there is a net saving in cost.

The semijoin concept can be readily adapted to perform joins in distributed spatial databases. However, the conventional semijoin has to be extended for additional considerations that are peculiar to spatial databases. These considerations are:

- The conventional semijoin uses the distinct values of the joining attribute in S to minimize the transmission cost from S_{site} to R_{site} and the cost to evaluate the semijoin. However, as most spatial descriptions are intrinsically complex data types and represent irregular, nonoverlapping partitions of space, there is no direct equivalent to a projection on a single-value attribute where the joining attribute is a spatial description. Consider, for example, a relation representing soil types in a region. Each record then describes a polygonal area occupied by a certain soil type. These polygons are represented as arrays of the coordinates of their vertices. In natural resource databases, spatial descriptions are typically hundreds

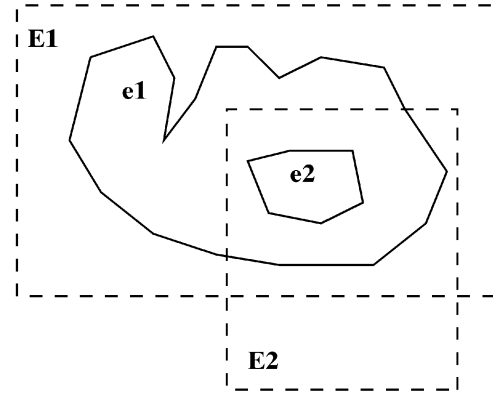


Fig. 4. E_1 overlaps E_2 , and e_1 includes e_2 .

to thousands of bytes long [9]. For cadastral databases, descriptions are much smaller, but still in the region of 100 bytes [2]. Therefore, transmitting the spatial descriptions remains costly.

- Evaluation of spatial relationships, such as containment, intersection, and adjacency between two polygons, is complex and expensive compared to testing equality of two single-value attributes. For example, our study shows that testing intersection of two polygons each with an average of six vertices costs 250 microseconds on a SUN SPARC-10 workstation, while the equality test of two single-value attributes is of the order of 0.1 microseconds.

To cut down on the transmission cost and local processing cost to evaluate the semijoin on the spatial attributes of R and S , we propose that approximations to the objects in R and S and a computationally less expensive, but weaker, spatial relationship be used. The approximations and the weaker spatial relationship must satisfy the *property of conservation*, i.e., for two objects that are spatially related, their approximations must also be related by the weaker relationship. For example, as shown in Fig. 4, an object e_2 is possibly contained in object e_1 only if its approximation E_2 intersects with e_1 's approximation E_1 .

To some extent, the idea of spatial semijoins using weaker relationships is similar to joins on approximations in [12]. Table 1 lists some examples of spatial relationships (denoted θ) and their approximations (denoted Θ).

Using approximations and a weaker relationship are motivated by two observations. First, the approximation is shorter than the full descriptions of the spatial object and, hence, will incur lower transmission costs. Second, geometrically-simple approximations, such as the minimum bounding rectangles, allow simple evaluation of spatial relationships and so reduce the computation cost when evaluating the semijoin. For example, referring to Fig. 4 again, transmitting rectangles with two vertices is definitely cheaper than transmitting the irregularly shaped objects. Moreover, checking for rectangle intersection is also cheaper than checking for polygon containment.

More formally, the *spatial semijoin* is defined as follows:

Definition 1. Let R and S be two spatial relations, and A and B be attributes of R and S on spatial domains, respectively. The

TABLE 1
Some Examples of Operations and their Approximations

$e_1 \theta e_2$	$E_1 \Theta E_2$
e_1 within distance d from e_2 (measured between center points)	E_1 within distance d from E_2 (measured between closest points)
e_1 overlaps e_2	E_1 overlaps E_2
e_1 includes e_2	E_1 overlaps E_2
e_1 contained in e_2	E_1 overlaps E_2

ϑ spatial semijoin between R and S on attributes A and B , denoted $R' = R \overset{s}{\ltimes}_{A\vartheta B}$, is defined by

$$R' = \{t \in R \mid \exists s \in S \text{ where } f_R(t.A) \varphi f_S(s.B)\}.$$

Here, ϑ and φ ($= g(\vartheta)$) are spatial operators, f_R and f_S are approximation functions, and g is a mapping function on relationships such that the following holds: For two records $t \in R$, and $s \in S$, $t.A \vartheta s.B$ is true implies $f_R(t.A) \varphi f_S(s.B)$ is true.

We have the following remarks on the definition of spatial semijoin.

1. Approximation functions are used to map the complex spatial descriptions of objects to a simpler form. The functions f_R and f_S may be different. For example, f_R may map each record of R to its *minimum bounding rectangle*, while f_S may map each record of S to its *rotated minimum bounding rectangle*.
2. The semantics of φ is dependent on ϑ . In fact, φ is a weaker relationship than ϑ . While ϑ and φ generally refer to the same relationship, they may be different. Table 1 illustrates some of these.
3. As a result of point 2, i.e., using a weaker relationship, the result of the spatial semijoin *contains* all the records of R that will participate in the final join operation. On the contrary to the conventional semijoin, where the result of the semijoin is the set of records of R that will participate in the final join operation, the spatial semijoin using a weaker relationship does not eliminate totally all records that do not contribute to the final answer. In other words, the spatial semijoin result contains both *hits* and *false drops*. Hits are records of R that will satisfy the join operation, ϑ , while *false drops* are those that satisfy the join operation φ but not ϑ .

3.2 The Framework

A distributed spatial join can be preprocessed using a spatial semijoin. The basic framework of a spatial join, $R \overset{s}{\bowtie}_{A\vartheta B} S$ at site S_{site} , using a semijoin can be expressed as follows.

Framework 1. Distributed Spatial Join Processing Using Semijoin

Input: Two relations R and S with spatial attributes A and B , respectively.

Output: $R \overset{s}{\bowtie}_{A\vartheta B} S$.

1. $B^f = \{s.B \mid s \in S\}$;
2. $B^f = \{f_S(b) \mid b \in B^f\}$;

3. **Send** distinct values of B^f from S_{site} to R_{site} ;
4. **Reduce** R to $R' = \{t \in R \mid \exists b \in B^f \text{ such that } f_R(t.A) \varphi b \text{ where } \varphi = g(\vartheta)\}$;
5. **Send** R' to S_{site} ;
6. **Perform** $R' \overset{s}{\bowtie}_{A\vartheta B} S$

The performance of the framework hinges on the approximations to the spatial descriptions of S , i.e., Step 2. We restrict our discussion here to three possible approximation functions:

- f_S is an identity function \mathcal{I} such that $\forall t \in B, \mathcal{I}(t) = t$. In other words, the spatial descriptions of the records of S are sent to the R_{site} .
- f_S is a 1-1 mapping such that each record of B^f is mapped to a record in B^f . However, instead of the complex spatial description, a simpler approximation that bounds the spatial object is used. For example, the minimum bounding rectangles (MBR) of the records may be used and transmitted.
- f_S is a m-1 mapping such that several records of B^f are mapped to the same B^f record. For example, a smaller set of MBRs can be used such that each MBR bounds several spatial objects.

The first two approaches lead to B^f having as many records as B . The last approach, on the other hand, has the potential for varying the size of B^f to optimize the total cost of the operation. The first approach has no false drops in R' if $\varphi = \vartheta$, while the last approach has the highest number of false drops in R' . However, the first approach incurs the highest cost in transmitting B^f to site R_{site} , while the last approach requires minimum transmission cost. Since the number of objects being reduced by S using the third approach might not be significant, transmitting R' could still be very expensive. Such an effect will defeat the purpose of a spatial semijoin.

The second and third approaches follow closely the usual practice in spatial database of structuring operations as a filter operation using simplified descriptions (or approximations) followed by full evaluation using the full descriptions of objects [23], [9], [8], [11], [36], [35]. Typically, a polygonal object is represented for the filter operation by its MBR because MBRs are inexpensive to store (16 bytes for single-precision) and spatial relationships between them are inexpensive to evaluate. The filter test is formulated not to reject cases that satisfy the full evaluation. However, it typically will not reject all cases that do not satisfy the full test. Design then seeks a good compromise between the false drops, the costs of the filter operation, and the costs of storing, and (if necessary) deriving the approximations. In

place of MBRs, more accurate single object filters, such as *convex hull*, *n*-corner, *g*-degree rotated *x*-range and *g*-degree rotated *y*-range [8], [36], [35], can be used to reduce the number of false drops. However, such filters require both additional storage and computation overhead (as more points are needed to represent the bounding box).

From the definition of the spatial semijoin and the description of the basic distributed join framework, we note that they are independent of the approximations used and the spatial relationships on the joining attributes of *R* and *S*. Thus, adopting different approximations and spatial relationships will lead to different families and classes of semijoin-based algorithms. Without loss of generality, we shall adopt the MBR as the basic approximation and intersection as the spatial relationship.

4 DISTRIBUTED SPATIAL JOIN ALGORITHMS

In this section, we present several distributed intersection-join algorithms that use spatial semijoins. We assume both *R* and *S* are indexed. This is not unreasonable since large spatial relations in practice usually have existing spatial indexes. The algorithms exploit existing spatial indexes in two ways:

- to provide the approximations for spatial objects;
- to perform the semijoin and final join using the indexes at the respective site.

4.1 Semijoins Using Multidimensional Approximations

For this category, we adopt R-tree as the indexing technique. When an R-tree index on *S* exists, it can be exploited for distributed spatial join processing. Since the MBRs held at each level of the R-tree form a candidate set of containing rectangles of the objects in *S*, the MBRs can be considered as approximations for objects in *S*. In other words, the MBRs at any level of the R-tree correspond to the result of an implicit *m*-1 mapping function on the objects of *S*. In fact, this implicit *m*-1 mapping function is a reasonably good one since data objects bounded by an MBR of an R-tree are clustered. Choosing the level of nodes in the R-tree to supply the MBRs is equivalent to choosing the number of MBRs approximating *S*. While using a higher level in the R-tree provides a smaller number of MBRs, employing a lower level provides better approximations.

The semijoin at R_{site} , and the final join at S_{site} are both local (single-site) spatial join operations. Since both the semijoin and final join are performed in a similar manner, we shall only describe how the semijoin is performed. The main concern here is that we do not have an index on the approximations of *S*, *S'*, at R_{site} . Two possible techniques can be adopted:

- Create an R-tree index for *S'* at R_{site} . (We have assumed that *R* is indexed using the same family of indexes, i.e., a R-tree in this case. Note that this is not a restriction on our proposed semijoin-based join techniques. If *R* is indexed using other indexing schemes such as the R^* -tree [7], we can also create an R^* -tree for *S'* and perform the join

accordingly.) This allows us to exploit existing join techniques proposed in [9] which require precomputed R-tree indexes to exist on both data sets. The join is then performed as described in Section 2.1.1.

- Employ a nested-index technique by performing a series of selections on *R*. This technique essentially treats each MBR of *S'* as a query window on *R*.

Our preliminary experimental study indicated that creating an R-tree is a very expensive operation. Using the data sets that we have, the cost of creating R-trees for the semijoin and final join is more than half the total processing cost using the second approach. As such, as well as for simplicity, we employ the nested-index technique in this paper.

4.2 Semijoins Using Linearized Single-Dimensional Object Mapping

For this category, we employ the quad-tree based key assignment to order spatial objects. For each object, the locational keys of its MBR are used as approximations.

The distributed spatial join can be performed by transmitting the locational keys. Since the locational keys are sorted, the semijoin at R_{site} and the final join at S_{site} can be performed using a merge-like algorithm, as described in Section 2.1.2. As each object has multiple keys, the join result may contain duplicates. Hashing is used to remove the duplicates.

We note that the semijoin is less complex than the final join. Since the join is a nonequijoin, the scanning of the sorted lists for the final join requires "backing up." On the other hand, the semijoin requires a full scan of *R* and, at most, a full scan of *S*. This is because it suffices to check that a record of *R* matches at least one record of *S'*, rather than multiple records of *S'*.

For the final join, to cut down the cost of I/O when records are backed-up, we cached those records of *R'* (and *S*) that join with multiple records of *S* (and *R'*).

5 PERFORMANCE STUDY

A performance study on the distributed join algorithms was performed to answer the following questions:

- How effective are the semijoin-based algorithms compared to the naive methods of evaluating a join on distributed spatial databases?
- What approximations will lead to the best performance?
- What is the relative performance between the algorithms that are based on approximations on cluster of objects and those that are based on single-dimensional locational keys?

In this section, we report the experiments conducted and their results.

5.1 The Experimental Setup

A data set of the land parcels for the whole State of South Australia was used for the experiments. This database has 762,000 records with polygonal spatial descriptions of an average of six vertices. Three pairs of test sets were extracted: sets 10R and 10S with 10,000 parcels each, 50R

TABLE 2
Properties of the R-Trees

	Relation R			Relation S		
	10K	50K	100K	10K	50K	100K
Cardinality						
height of the tree	3	3	4	3	3	4
number of nodes	285	1415	2829	281	1397	2838

and 50S with 50,000 parcels, and 100R and 100S with 100,000 parcels each. The generation of pairs of sets sought to ensure a controlled number of intersecting parcels. The database was initially divided into three parts corresponding to three geographic regions of South Australia, so that the upper part contained 280,000 parcels, the middle 245,000 parcels, and the lower 237,000 parcels. The set 10R was generated by randomly selecting two thirds of the records (i.e., 6,666 parcels) from the upper part and one third (i.e., 3,334 parcels) from the middle. The set 10S was generated by selecting one third of the records from the middle part and two thirds from the lower. The objects of S are translated by 100m northward and eastward. In this way, the objects in the database that appear in both R and S become “different.” The other pairs of test sets were similarly generated. Tests showed that there were 135 intersections between objects in the pair 10R and 10S, 3,253 in the pair 50R and 50S, and 13,096 in the pair 100R and 100S.

The performance of the various algorithms were compared on the total time for the distributed spatial join processing. We omit the cost of producing the final result since it is the same for all strategies. The total cost to perform a distributed spatial join is given as:

$$C_{total} = C_{transmit} + C_{cpu} + C_{io},$$

where $C_{transmit}$, C_{cpu} , and C_{io} , respectively, represent the transmission, CPU, and I/O cost required for the join.

The transmission cost incurred includes the cost of transmitting the approximations of S , S' , from S_{site} to R_{site} , and the cost of transmitting records of R that will participate in the final join, R' , from R_{site} to S_{site} .

The CPU cost comprises several components. These include the cost to initiate I/Os, to initiate sending and receiving of objects/approximations, to extract the approximations, to perform the semijoin at R_{site} , and to perform the final join at S_{site} .

The I/O cost comes from fetching records for generating S' at S_{site} , storing S' at R_{site} , fetching S' for and performing the semijoin, fetching R' , storing R' at S_{site} , refetching R' for and performing the final join.

The various algorithms were implemented on a SUN SPARC-10 machine. The evaluation of the algorithms was conducted by performing the semijoin and the final join using the data sets. These provided information on the size of the approximations used to evaluate the semijoin and the size of the semijoin result. We also monitored the CPU usage and the number of page accesses. While the value of the CPU cost monitored reflects the CPU usage, the transmission and I/O cost are computed, respectively, as follows:

$$C_{transmit} = \text{number of bytes transmitted} \cdot \frac{1}{\omega_{bandwidth}}$$

$$C_{io} = \text{number of page accesses} \cdot io,$$

where $w_{bandwidth}$ represents the bandwidth of the communication network and io represents the cost per page access.

In our study, unless otherwise stated, $\omega_{bandwidth}$ is 100KBytes/sec, the I/O cost for a 8 KBytes block is 0.025 sec/block, and the system has a buffer size of 10 MBytes. Both R-trees and B⁺-trees of locational keys were implemented in C.

Two series of experiments were conducted. The first evaluates the algorithms that make use of approximations obtained from the R-tree, and the second examines the performance of the locational key algorithms. We also conducted an experimental study on the relative performance of the R-tree based and locational key based algorithms.

5.2 Results for Algorithms Using Multidimensional Approximations

The R-tree index node is 2KBytes. Each node has at most 56 entries and at least 28 entries. The characteristics of the trees are summarized in Table 2.

Following the distributed join strategies described in Section 4.1, we studied the following three strategies.

- **Algorithm RT-N**. This is the naive algorithm that transmits R to S_{site} , and evaluates the join directly as a sequence of selections on S .
- **Algorithm RT-L**. The MBR of each object of S is used as its approximation, and these MBRs are taken from the leaf nodes of the R-tree for S . The MBRs are transmitted to R_{site} to reduce R before R is sent to S_{site} for the final join.
- **Algorithm RT-I**. This algorithm is similar to algorithm RT-L except the approximations of S are taken from the internal nodes at the level immediately above the leaf level of the R-tree for S .

Fig. 5 shows the result of the experiment when both relations R and S are of the same size. In the figure, the lower, middle, and upper bars represent the CPU, transmission, and I/O cost, respectively. The corresponding information on the number of objects transmitted for the semijoin-based algorithms are tabulated in Table 3. In Table 3, S' denotes the number of approximations (i.e., MBRs) of S transmitted to R_{site} for the semijoin, and R' is the number of R objects satisfying the semijoin and to be transmitted to S_{site} for the final join.

A number of observations can be made, bearing in mind they are valid only within the context of the characteristics

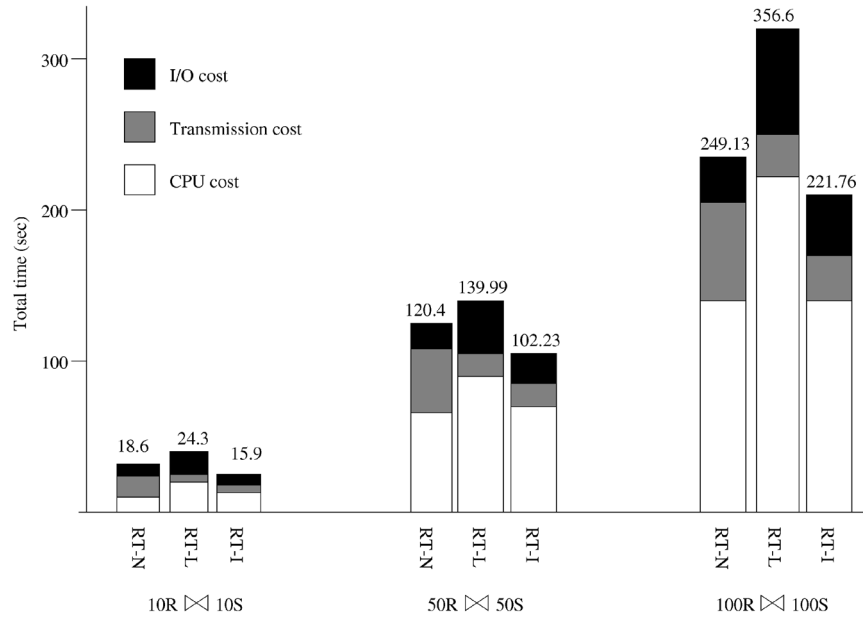


Fig. 5. Comparisons of R-tree based algorithms.

of the data sets used (particularly the short spatial descriptions of the Land Information Systems data) and of the relatively high transmission rate used. First, for all the algorithms, the local processing cost ($C_{cpu} + C_{io}$) is significantly higher than the communication cost. This confirms that local processing cost cannot be ignored during query processing for distributed spatial joins. We note that several recent join algorithms [5], [15], [18], [27] can be employed for local join processing. While these techniques may reduce the CPU and I/O cost, we expect the relative performance between the various schemes to remain.

Second, the result shows the effect of choosing the number of MBRs to approximate S . There is a tradeoff between the number of MBRs used to approximate S and the number of false drops. As shown in Table 3, a large number of MBRs results in a small number of false drops, and a small number of MBRs leads to a large number of false drops. From Fig. 5, we see that algorithm RT-L, which uses a large number of MBRs to approximate S , performs worse than algorithm RT-N. RT-L managed to reduce the communication cost through the semijoin but its processing cost is much larger than those of RT-N. This is because RT-L requires as many R-tree probes to perform the semijoin as RT-N takes to perform the final join. The additional overhead incurred for the final join is more than enough to offset the benefits of transmitting the reduced R . On the other hand, algorithm RT-I outperforms algorithm RT-N.

Transmitting a smaller number of MBRs leads to lower communication and processing cost for the semijoin. However, a larger number of false drops results in higher communication and processing cost for the final join. It turns out that the total cost is reduced. The particular significance of this observation is that the containing rectangles of the internal nodes of an R-tree one level above the leaf nodes is a good source of approximating rectangles to the objects of the indexed relation. This means that a semijoin algorithm can reuse the spatial index normally recommended for large spatial relations.

To study the effect of the size of the joining relations have on the algorithms, we conducted experiments using relations of different sizes. The results are summarized in Fig. 6. Several interesting observations can be made. First, when R is large as compared to S (see the cases for 100R x 10S and 50R x 10S), RT-L performs best. When joining a large R with a small S , the effect of false drops becomes significant. For algorithms RT-N and RT-I, the large number of false drops leads to high communication cost and high processing cost for the final join. Second, algorithm RT-I outperforms RT-N in all cases. This shows that semijoin algorithms are effective for distributed spatial databases. Before leaving this section, we would like to point out that we have not exploited the obvious optimization of treating R as the larger of the relations. While this may help in some cases, it is not expected to be so in our context as the result site is fixed at S_{site} (which means an additional phase of shipping the result from R_{site} to S_{site} would be necessary if the optimization is adopted for small R and large S relation pairs).

To understand more fully the effects of transmission rates on the algorithms, we also performed sensitivity studies on the transmission rates. Figs. 7a, 7b, and 7c show the results of these tests when the communication

TABLE 3
Number of Objects Transmitted for R-Tree Based Algorithms

Cardinality	RT-L		RT-I	
	S'	R'	S'	R'
10K/10K	10000	231	272	3329
50K/50K	50000	4883	1357	16651
100K/100K	100000	17112	2759	33319

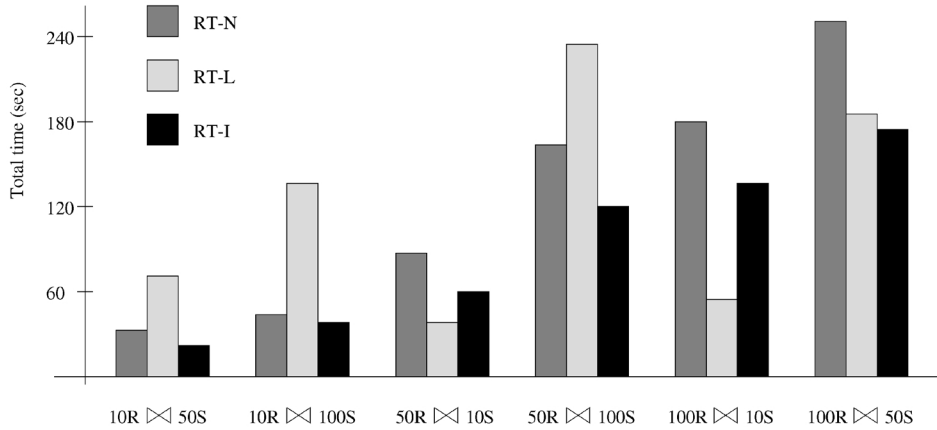


Fig. 6. R-tree based joins of different relation sizes.

bandwidth varies from 20 KBytes/sec to 100 KBytes/sec, and Fig. 7d shows the result for $100R \bowtie 100S$ when the communication bandwidth varies from 100 KBytes/sec to 1 MBytes/sec. This effectively models the range of network speeds from WAN to LAN. In all the experiments, the length of spatial description is fixed at 48 bytes. The result shows that when the communication bandwidth is low (< 30 KBytes/sec), both the semijoin-based algorithms are more efficient than the naive approach. At low communication bandwidth, transmission cost may become a dominant component in the total processing cost. The ability to prune away objects that do not satisfy the join conditions makes the semijoin-based algorithms effective and attractive. On the other hand, the transmission cost for the naive strategy to transmit the entirety of R from R_{site} to S_{site} is very high, resulting in its poor performance. However, as the bandwidth increases, the naive strategy slowly catches up with the semijoin-based techniques. At bandwidth greater than 30 KBytes/sec, it outperforms RT-L, and at bandwidth greater than 300 KBytes/sec, it performs best. This is because, at higher communication bandwidth, the communication cost decreases and the processing cost dominates performance. It turns out that the semijoin-based algorithms incur higher processing cost for small spatial descriptions. This is because the semijoin-based algorithms need to write out the approximations at R_{site} and S_{site} , and the result of the semijoin at S_{site} . Moreover, most of these I/Os are incurred twice—one to write, and the other to reread. We see a tradeoff between the total I/O cost incurred and the I/O cost saved by the semijoin-based algorithms. Note that the I/O cost saved comes from two components: the number of false drops reduced by the algorithm (the naive method did not reduce any false drops), and the size of the spatial descriptions. When the length of spatial descriptions is small, the I/O cost saved is relatively small.

The length of spatial descriptions affects both the I/O and communication cost. Figs. 8a, 8b, and 8c show the results as we vary the length of the spatial descriptions from 50 bytes to 1 KBytes. The lower end of the range (towards 50 bytes) models LIS applications with their small spatial descriptions while the higher end (towards 1 KBytes)

represents GIS applications with long descriptions. In this study, the communication bandwidth of 100 KBytes/sec was used. The result shows that RT-I always outperforms RT-N, regardless of the size of the spatial descriptions. RT-L outperforms RT-N for large spatial descriptions. This is due to the significantly higher transmission cost and I/O cost for RT-N. Comparing RT-I and RT-L, we note that RT-L outperforms RT-I for large spatial descriptions. As the size of spatial descriptions increases, false drops will result in more data being transmitted leading to higher communication and I/O cost and, hence, RT-L becomes comparable to RT-I.

From the experiments, there is no doubt about the effectiveness of using semijoins since RT-N is always worse than either RT-I or RT-L (except for very high communication bandwidth). However, the choice of the approximations is critical to the effectiveness of a semijoin. For applications with small spatial descriptions, RT-I is the best choice. On the other hand, for applications with large spatial descriptions, RT-L proves to be effective. The results are expected because when the spatial description is small, unless the reduced R is very small, the saving will not be significant. In this case, the overhead for transmitting the (reducing) approximations of S should not be too high for semijoin-based algorithms to be effective. As such, RT-I, which has a smaller number of approximations, is the better scheme. When the spatial description is large, the saving from transmitting spatial objects that are of no interest to the answer is much higher and more accurate approximations are more effective and the cost of transmitting these approximations are much lower than the cost of transmitting real objects.

Before leaving this section, we would like to add that the data set that we used for land parcels is densely populated. For sparsely populated data sets, it may be possible for RT-L to outperform RT-I since the MBRs at RT-I are likely to generate a large number of false drops.

5.3 Results for Algorithms Using Single Dimensional Object Mapping

In this section, we report on the experiments and results for the algorithms based on single-dimensional object mapping. We note that most of the results in this section have

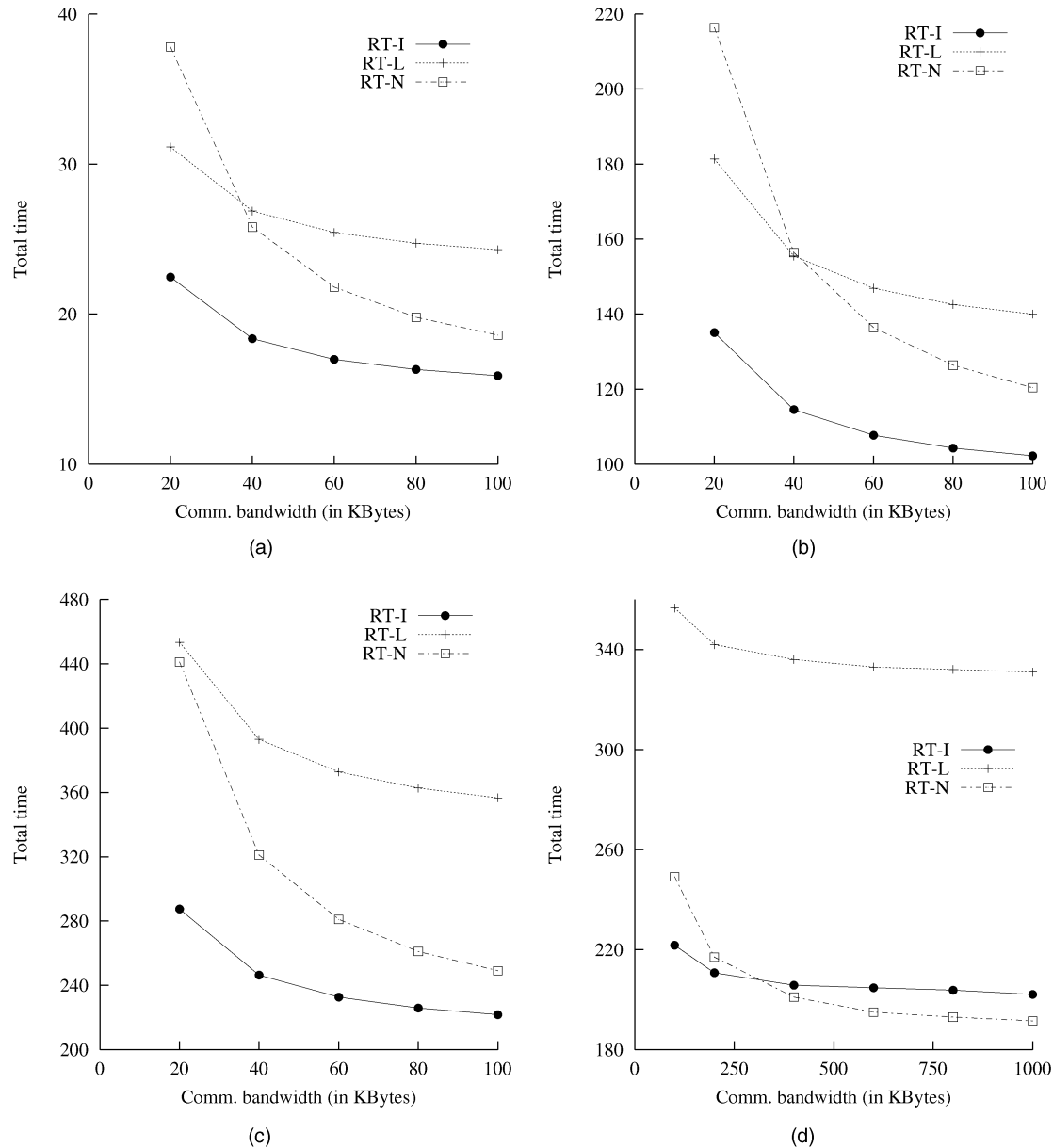


Fig. 7. R-tree based joins with varying communication bandwidth. (a) 10R \bowtie 10S. (b) 50R \bowtie 50S. (c) 100R \bowtie 100S. (d) 100R \bowtie 100S.

also been reported in [4]. From the data sets, we generated the locational key values of the objects. Three strategies based on locational keys were evaluated. They are:

- **Algorithm QT-N.** This is the naive algorithm which transmits R and the associated locational keys to S_{site} , and evaluates the join directly. (We have conducted preliminary tests on a nested-loops algorithm that transmits R to S_{site} , and performs the join as a series of spatial selections of R records on S . The results turned out to be bad. It is always worse than algorithm QT-N, and performed as much as four times worse than algorithm QT-N. As such, we omitted the nested-loops algorithm in our experiments.)
- **Algorithm QT-4.** This algorithm uses a semijoin. The locational keys of all objects in S are used as the approximations. These locational keys are readily

available from the leaf nodes of the B^+ tree for S . In our study, each object has at most four locational keys. (We can derive a family of algorithms with different number of locational keys. Our preliminary study, however, shows that four locational keys perform well.) Duplicate keys are removed before transmission.

- **Algorithm QT-C.** In algorithm QT-4, there may be redundancy in the approximations of S . First, a region represented by a locational key may be contained in a region represented by another locational key. Second, four regions represented by four distinct locational keys may be merged into a bigger region represented by a single locational key. To remove such redundancy, the approximations of S are *compacted* before transmitting to R_{site} . We denote the new algorithm QT-C. Note that compaction does not result in any loss of information. While

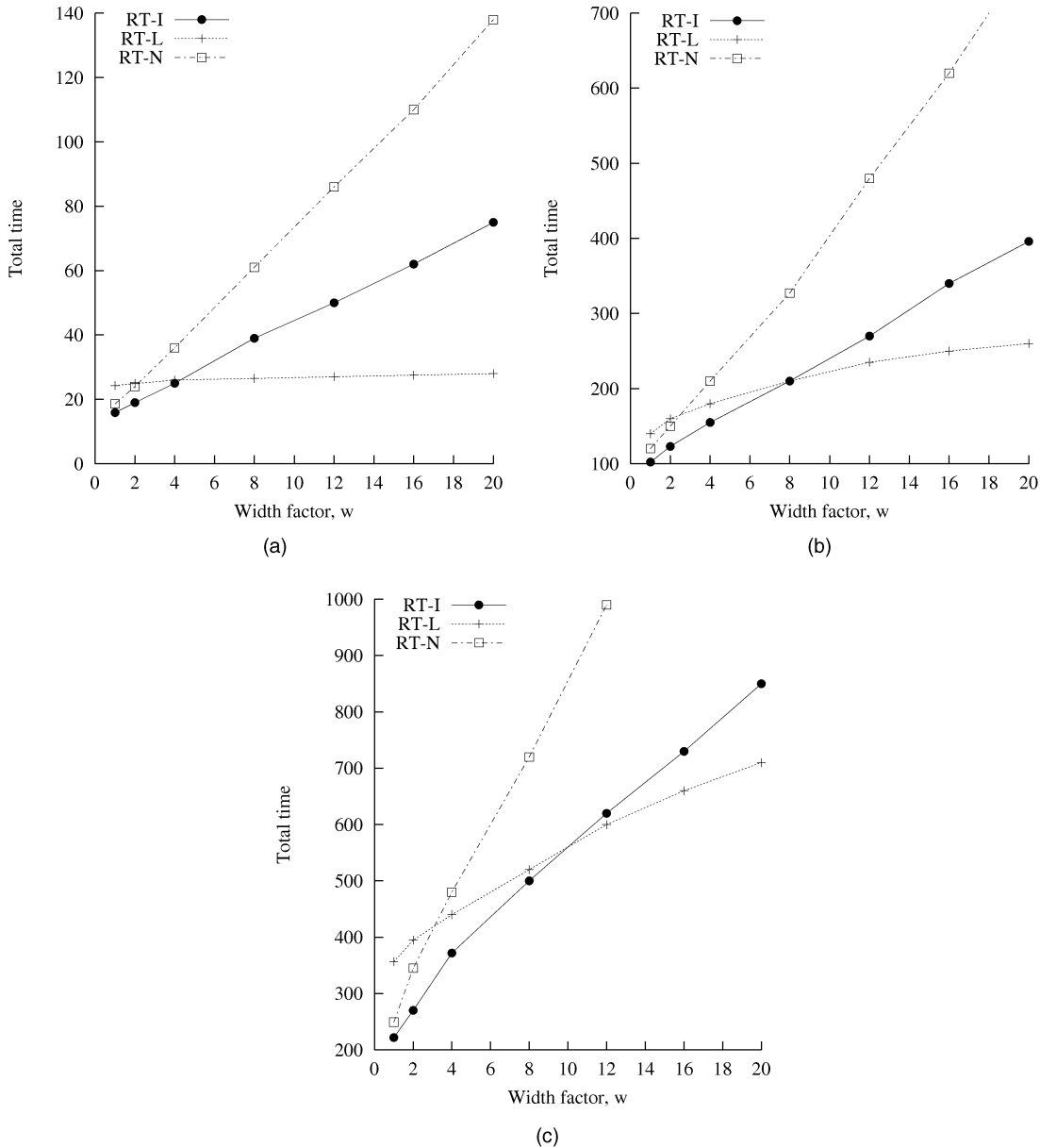


Fig. 8. R-tree based joins with varying record width. (a) Record width = 50w bytes ($10R \approx 10S$). (b) Record width = 50w bytes ($50R \approx 50S$). (c) Record width = 50w bytes ($100R \approx 100S$).

compaction minimizes the size of the approximations to be transmitted and the cost of the semijoin, it also incurs additional CPU cost to compact the approximations.

With the data sets that we have, each object has an average of 2.3 locational keys, so the 10K relations have about 23K locational keys. Table 4 summarizes the data for the various relations used.

We repeated the same sets of experiments on the multidimensional approximation based algorithms using the locational key algorithms. The results are summarized in Figs. 9, 10, 11. In Fig. 9, both relations R and S are of the same size. The corresponding information on the data transmitted are shown in Table 5. Here, S' corresponds to the number of locational keys of S transmitted to R_{site} , R' is the cardinality of the semijoin result (i.e., number of objects of R that satisfy the semijoin), and R'' is the corresponding

number of locational keys transmitted for the semijoin result.

First, we observe that, unlike the multidimensional approximation based algorithms, the communication cost for locational key based algorithms is more significant. Two reasons account for this—the lower local processing (especially CPU) cost and the larger amount of data to be transmitted. By using a sort-merge join method, the two lists of locational keys can be scanned simultaneously, resulting in low CPU cost for the semijoin and final join. On the other hand, the size of the approximations could be large, and transmitting the approximations becomes costly.

Second, we note that the algorithms based on semijoins outperform the naive approach. For algorithm QT-N, the communication cost dominates performance. Algorithm QT-4 performs better since it is able to cut down the size of the data to

TABLE 4
Total Number of Locational Keys

Cardinality	Relation R			Relation S		
	10K	50K	100K	10K	50K	100K
Total number of keys	23189	115484	231426	23175	115904	232169

be transmitted. Algorithm QT-C outperforms QT-4 slightly because it reduces the size of the approximations to be transmitted. From the result, we see that QT-C is more effective for large S . This is because for large S , more approximations can be compacted.

Fig. 10 shows the results of the experiments when R and S have different relation sizes. Both algorithms QT-4 and QT-C are inferior to QT-N when S is larger than R . This is expected since for large S , transmitting the large number of approximations of S to R_{site} is not less expensive than transmitting the entirety of the smaller R to S_{site} . Thus, the savings gained in reducing R cannot outweigh the overhead.

Figs. 11a, 11b, and 11c show the results for spatial joins of different relation sizes when the communication bandwidth varies from 20 KBytes/sec to 100 KBytes/sec. The result as communication bandwidth varies from 100 KBytes/sec to 1 MBytes/sec is shown in Fig. 11d. As shown in the figures, the semijoin-based algorithms are more effective at lower communication bandwidth (< 300 KBytes/sec). This is expected since communication cost is critical in locational key based algorithms as more data are transmitted.

In Figs. 12a, 12b, and 12c, we note that the semijoin-based algorithms also perform best for varying length of spatial descriptions. The longer the spatial descriptions, the higher the communication cost (and I/O cost). This leads to the poor performance of the naive methods. Note that the difference between the two semijoin algorithms is not significant. In fact, the difference decreases as the length increases. The compaction of locational keys only benefits the transmission of the approximations which do not change with the length of the spatial descriptions, i.e., the total cost is increasing while the saving due to compaction stays constant.

5.4 Comparison of Multidimensional and Single-Dimensional Approximations Algorithms

A fair comparison of multidimensional approximations and single-dimensional algorithms should include a comprehensive study on

1. a set of operations (e.g., selection, intersection-join, adjacency-join, etc.),
2. a large set of data, and
3. different applications (e.g., GIS, LIS).

We would thus note that our comparative study between the R-tree and locational key algorithms reported in this

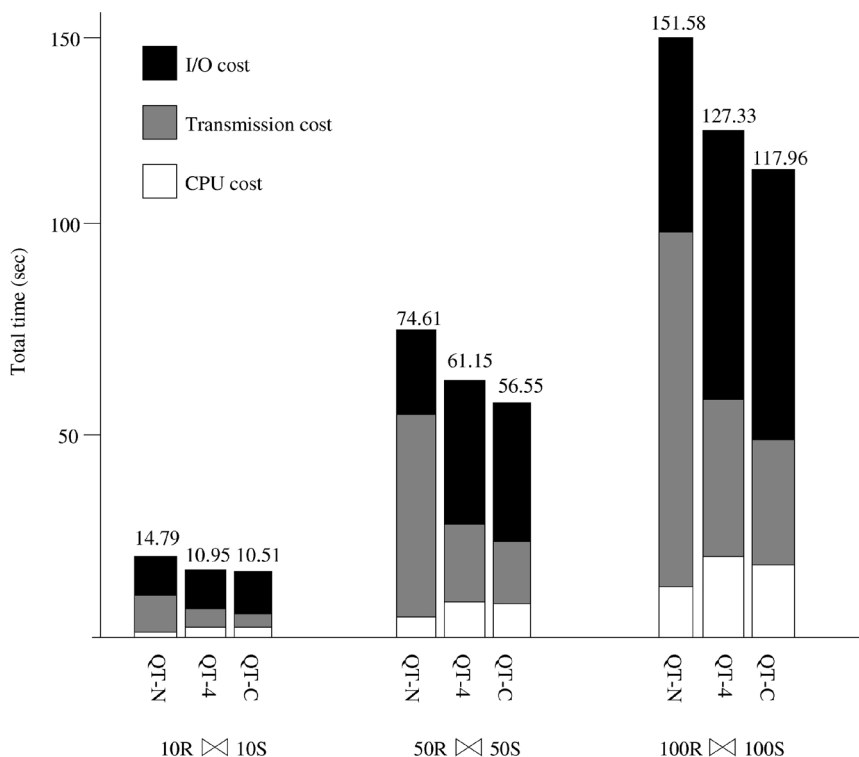


Fig. 9. Comparisons of locational key based algorithms.

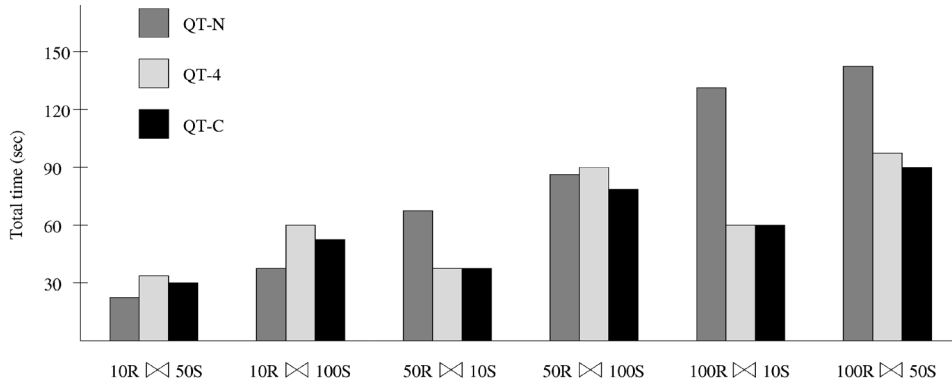


Fig. 10. Locational keys based joins of different relation sizes.

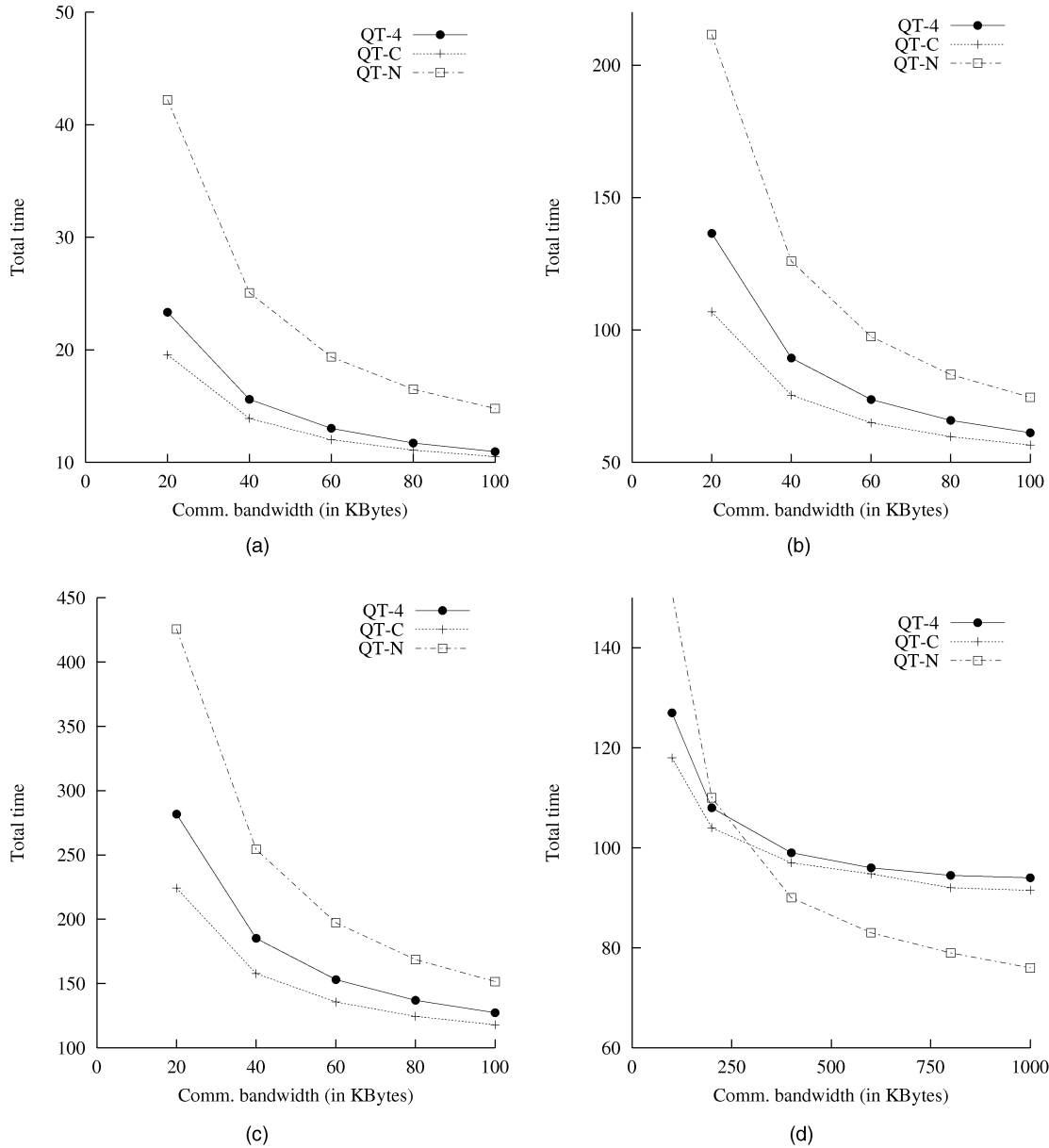


Fig. 11. Locational keys based joins with varying communication bandwidth. (a) 10R x 10S. (b) 50R x 50S. (c) 100R x 100S. (d) 100R x 100S.

section is valid only with respect to our data sets. Furthermore, we have restricted our evaluation to only the intersection-join operation.

Based on the results of the experiments on the R-tree based and locational keys based algorithms, we compare RT-I with QT-C. For simplicity, we denote these two

TABLE 5
Number of Objects Transmitted for Locational Key Based Algorithms

Cardinality	QT-4			QT-C		
	S'	R''	R'	S'	R''	R'
10K/10K	22045	4113	2011	11571	4113	2011
50K/50K	92385	36892	16977	14188	36892	16977
100K/100K	158470	86532	38005	8128	86532	38005

algorithms as RT and QT, respectively. Figs. 13, 14, 15, and 16 are some representative sets of results.

From the results, we note that algorithm QT outperforms RT in almost all cases. As seen in Fig. 13, algorithm RT is very costly in terms of local processing, especially the CPU cost in probing the R-tree. On the other hand, QT

requires only one scan of sorted lists of locational keys, which makes it very efficient. In the largest test data sets used (i.e., $100R \bowtie^s 100S$), a cache size of 20 pages (4KBytes each) is enough to contain all records of R' that join with multiple records of S . In other words, both relations need to be read only once.

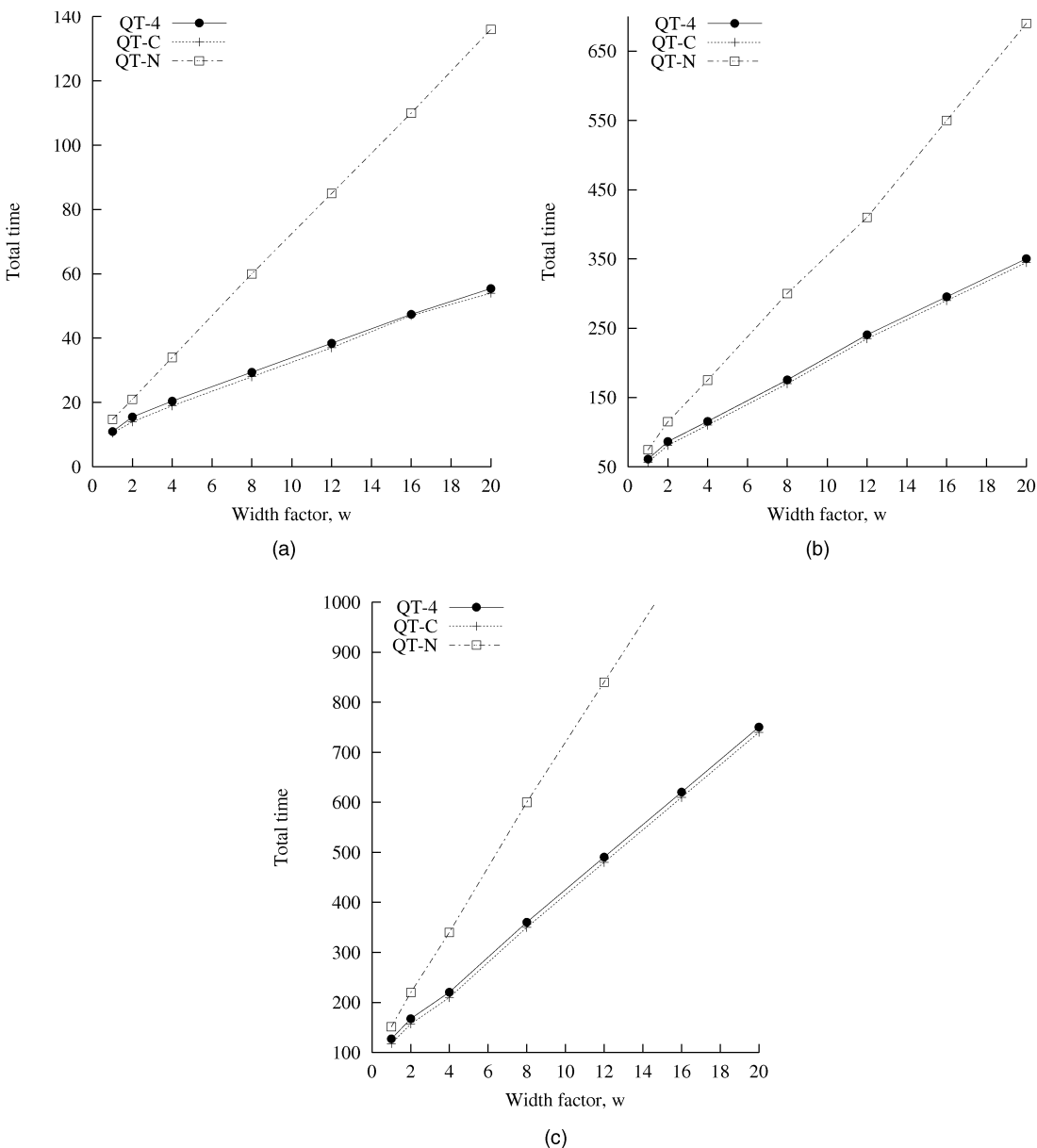


Fig. 12. Locational keys based joins with varying record width. (a) Record width = $50w$ bytes ($10R \bowtie^s 10S$). (b) Record width = $50w$ bytes ($50R \bowtie^s 50S$). (c) Record width = $50w$ bytes ($100R \bowtie^s 100S$).

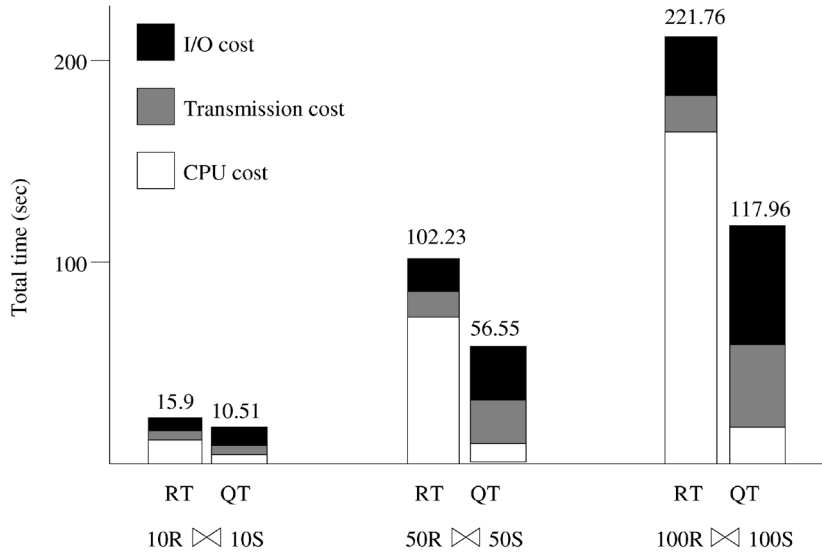


Fig. 13. Comparisons of R-tree and locational key based algorithms.

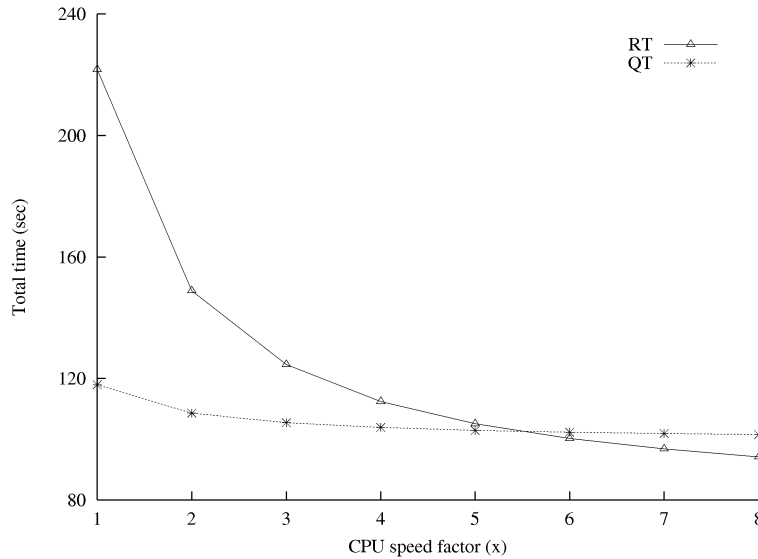


Fig. 14. Effect of CPU speed.

Because of the high CPU cost of RT, we decide to investigate the effect of faster CPU speed with fixed I/O and communication costs. Fig. 14 shows the result for 100R x 100S using the default setting. In the figure, the x -axis denotes the CPU speed factor, e.g., a value of 2 refers to a processor speed twice of that used in our study. As shown in the figure, with a faster processor, the RT can outperform QT. However, this is achieved only with very fast processors (factor > 6).

Fig. 15 shows the results as the communication bandwidth varies. In Fig. 15a, the spatial description is short (50 bytes). We note that while QT is superior for a wide range of bandwidth, the gain in performance over RT decreases as the bandwidth decreases. Moreover, when the spatial description is large (1,000 bytes), RT performs equally well as QT at low bandwidth (see Fig. 15b). In both cases, the reason is because QT transmits more data—both the locational keys and the data. Recall that the locational keys of an object is obtained from its MBR. This means that QT is less effective in reducing R during the semijoin, i.e.,

for QT, the number of false drops is larger. In our study, we find that the false drops produced by QT is twice as many as that produced by RT. As a result, more data is transmitted and more I/Os are incurred.

Fig. 16 compares the two algorithms for varying length of spatial descriptions. Again, the study was conducted on both low and high communication bandwidth (20 and 100 KB/sec, respectively). The results confirm our earlier observation that QT is superior for high communication bandwidth and small spatial descriptions. On the other hand, RT performs better at low communication bandwidth and large spatial descriptions.

To summarize, the results show that RT and QT perform equally well for GIS applications. For LIS applications, QT is best for LIS applications. However, with faster CPU, RT can be promising.

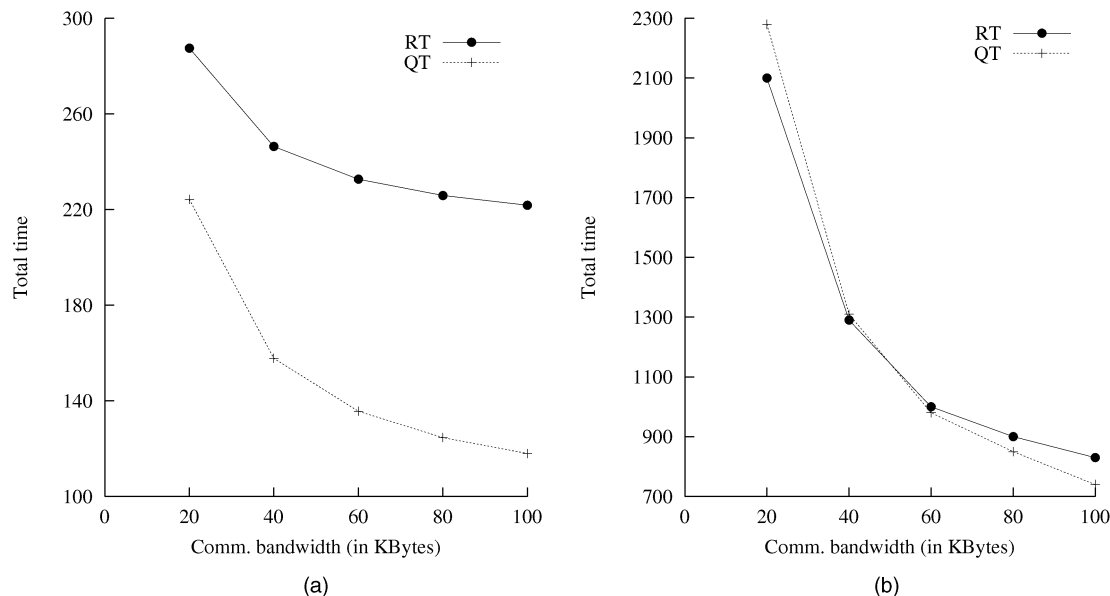


Fig. 15. Varied communication bandwidth for small and large spatial descriptions. (a) Record width = 50 bytes. (b) Record width = 1,000 bytes.

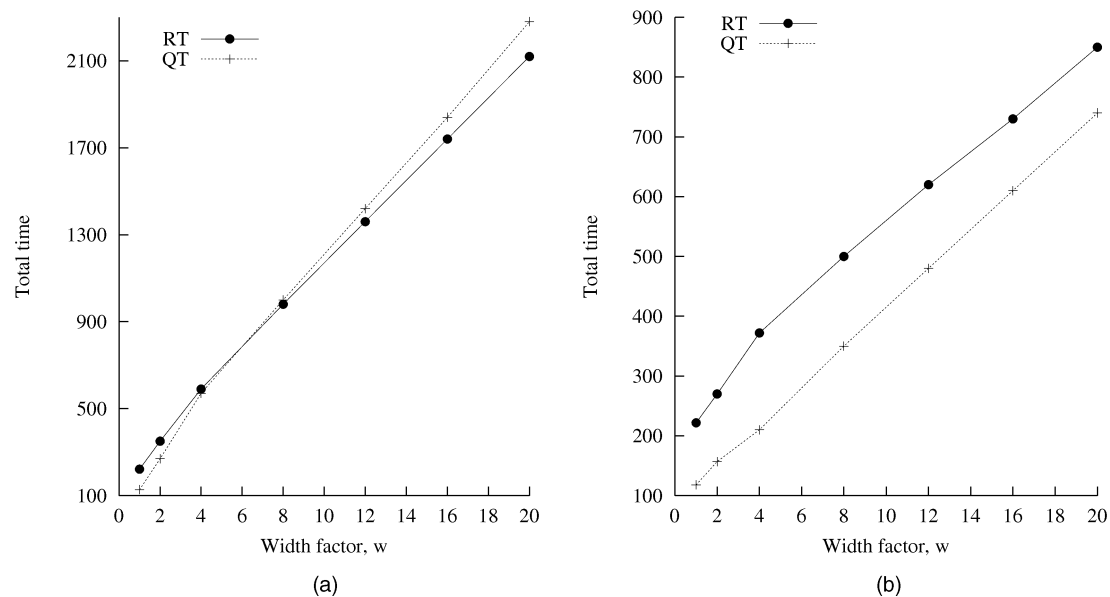


Fig. 16. Varied length of spatial descriptions for low and high communication bandwidth. (a) Communication bandwidth = 20 KB/sec. (b) Communication bandwidth = 100 KB/sec.

6 CONCLUSIONS

In this paper, we have presented the spatial semijoin operator as a basis for improved algorithms for evaluation of joins on distributed spatial databases. Its formulation and application draw on the concepts of the semijoin of conventional databases and of the filter test of spatial query processing. The application of a spatial semijoin in two families of algorithms, one based on the use of multidimensional approximations and the other based on the use of single-dimensional locational keys, has been described. The multidimensional approximation based semijoin algorithms use the minimum bounding rectangles derived from the nodes at a certain level of the R-tree as approximations. On the other hand, the single-dimensional locational key based algorithms exploit sort-merge

techniques by representing each object with at most four single-dimensional locational keys.

Our experimental study showed that semijoin algorithms provide useful reductions in the cost of evaluating a join in most cases. For the algorithms that are based on the approximations in R-tree, the choice of the number of approximations is critical for different domains of application. In particular, for Geographic Information System (GIS) applications, a larger number of approximations is preferred. However, in applications like Land Information System (LIS) applications, a smaller number of approximations result in better performance. For the locational key algorithms, while compacting large relations could reduce the cost, the gain is not significant, especially for applications with large spatial descriptions. The comparison between these two families of algorithms showed that both

locational key and multidimensional approximation based semijoin algorithms are suited for GIS applications, while the locational key based semijoin algorithms are more effective for LIS applications. However, with a faster CPU, we can expect the R-tree based methods to be promising.

We plan to extend this work in several directions. First, we would like to cut down the high CPU cost of the R-tree based algorithms. Second, we have not adequately addressed the issue of buffering. Recent results on the effect of buffering, and buffering schemes will provide a good starting point [16]. Finally, for the locational key based techniques, besides compaction which is nonloss, we plan to look at how approximations may impact the performance of the algorithm.

ACKNOWLEDGMENTS

The authors would like to thank Robert Power (of CSIRO) and Jeffrey X. Yu (of Dept of Computer Science, Australian National University) for their contributions in discussion. They have also read an earlier draft of this paper and provided helpful comments. Robert Power also helped set up the experiments for this work. The anonymous referees also provided insightful comments that help improve the paper.

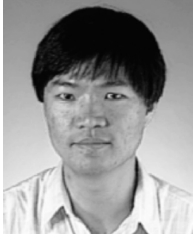
REFERENCES

- [1] D.J. Abel, "A b^+ -tree Structure for Large Quadrees," *Int'l J. Computer Vision, Graphics, and Image Processing*, vol. 27, pp. 19–31, 1983.
- [2] D.J. Abel, "Some Evolutionary Paths for Spatial Database," *Proc. Int'l Symp. Next Generation Database Systems and Applications*, pp. 1–10, 1993.
- [3] D.J. Abel, B.C. Ooi, R. Power, K.L. Tan, G. Williams, and X. Zhou, "The Virtual Database: A Tool for Migration from Legacy Lis," *Proc. Australasian Urban and Regional Information Systems Assoc. (AURISA' 94)*, pp. 117–126, 1994.
- [4] D.J. Abel, B.C. Ooi, K.L. Tan, R. Power, and J.X. Yu, "Spatial Join Strategies in Distributed Spatial Dbms," *Proc. Fourth Int'l Symp. Large Spatial Databases*, pp. 348–367, Aug. 1995.
- [5] L. Arge, O. Procopiuc, S. Ramaswamy, T. Suel, and J. Vitter, "Scalable Sweeping-Based Spatial Join," *Proc. Int'l Conf. Very Large Data Bases*, pp. 570–581, 1998.
- [6] P. Bernstein and D.M. Chiu, "Using Semi-Joins to Solve Relational Queries," *J. ACM*, vol. 28, 1981.
- [7] T. Brinkhoff, H. Kriegel, R. Schneider, and B. Seeger, "The r^+ -tree: An Efficient and Robust Access Method for Points and Rectangles," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 322–331, 1990.
- [8] T. Brinkhoff, H. Kriegel, R. Schneider, and B. Seeger, "Multi-Step Processing of Spatial Joins," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 197–208, 1994.
- [9] T. Brinkhoff, H. Kriegel, and B. Seeger, "Efficient Processing of Spatial Joins Using R-Trees," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 237–246, 1993.
- [10] J.L. Smith and D.J. Abel, "A Data Structure and Query Algorithm for a Database of a Real Entities," *Australian Computer J.*, vol. 16, no. 4, pp. 147–154, 1984.
- [11] V. Gaede and O. Gunther, "Processing Joins with User-Defined Functions," Technical Report TR-94-103, Institut für Wirtschaftsinformatik, Humboldt-Universität zu Berlin, 1994.
- [12] O. Gunther, "Efficient Computation of Spatial Joins," *Proc. IEEE Int'l Conf. Data Eng.*, pp. 50–59, 1993.
- [13] A. Guttman, "R-trees: A Dynamic Index Structure for Spatial Searching," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 47–57, 1984.
- [14] Y-W Huang, N. Jing, and E. A. Rundensteiner, "Spatial Joins Using R-Trees: Breadth-First Traversal with Global Optimizations," *Proc. Int'l Conf. Very Large Data Bases*, pp. 396–405, 1997.
- [15] N. Koudas and K.C. Sevcik, "Size Separation Spatial Joins," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 324–335, 1997.
- [16] S. Leutenegger and M. Lopez, "The Effect of Buffering on the Performance of R-Trees," *Proc. Intl Conf. Data Eng.*, pp. 164–171, 1998.
- [17] M-L. Lo and C.V. Ravishankar, "Spatial Joins Using Seeded Trees," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 209–220, 1994.
- [18] M-L. Lo and C.V. Ravishankar, "Spatial Hash-Joins," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 247–258, 1996.
- [19] H. Lu, R. Luo, and B.C. Ooi, "Spatial Joins by Precomputation of Approximation," *Proc. Sixth Australasian Database Conf.*, pp. 132–142, 1995.
- [20] W. Lu and J. Han, "Distance-Associated Join Indices for Spatial Range Search," *Proc. Ninth Int'l Conf. Data Eng.*, pp. 284–292, 1992.
- [21] G.M. Morton, "A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing," technical report, 1966.
- [22] J. Nievergelt, H. Hinterberger, and K.C. Sevcik, "The Grid File: An Adaptable, Symmetric Multikey File Structure," *ACM Trans. Database Systems*, vol. 9, no. 3, 1984.
- [23] J.A. Orenstein, "Spatial Query Processing in an Object-Oriented Database System," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 326–333, 1986.
- [24] J.A. Orenstein, "A Comparison of Spatial Query Processing Techniques for Naive and Parameter Spaces," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 343–352, 1990.
- [25] J.A. Orenstein, "An Algorithm for Computing the Overlay of K-Dimensional Spaces," *Proc. Second Int'l Symp. Large Spatial Databases*, pp. 381–400, 1991.
- [26] M.T. Ozsu and P. Valduriez, *Principles of Distributed Database Systems*. Prentice-Hall, 1991.
- [27] J.M. Patel and D. DeWitt, "Partition-Based Spatial Merge Join," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 259–270, 1996.
- [28] D. Rotem, "Spatial Join Indices," *Proc. Int'l Conf. Data Eng.*, pp. 500–509, 1991.
- [29] H. Samet, *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.
- [30] H.J. Schek, A.P. Sheth, B. D.Czejdo, eds., *Proc. Third Workshop Research Issues in Data Eng.: Interoperability in Multidatabase Systems*. IEEE CS Press, 1993.
- [31] T. Sellis, N. Roussopoulos, and C. Faloutsos, "The R^+ -tree: A Dynamic Index for Multi-Dimensional Objects," *Proc. Int'l. Conf. Very Large Data Bases*, pp. 507–518, 1987.
- [32] K.C. Sevcik and N. Koudas, "Filter Trees for Managing Spatial Data Over a Range of size Granularities," *Proc. Int'l Conf. Very Large Data Bases*, pp. 16–27, 1996.
- [33] Tomlinson Associates Ltd., *GIS Planning—Land Status and Assets Management*. Office of Geographic Data Coordination, 1993.
- [34] P. Valduriez, "Join Indices," *ACM Trans. Database Systems*, vol. 12, no. 2, 1987.
- [35] H.M. Veenhof, P.M.G. Apers, and M.A.W. Houtsma, "Optimization of N-Way Spatial Joins Using Filters," *Proc. 13th British Nat'l Conf. Databases*, 1995.
- [36] H.M. Veenhof, M.A.W. Houtsma, and P.M.G. Apers, "Query Optimization for Gis Using Filters," *Proc. ACM Workshop Advances in Geographic Information Systems*, 1993.
- [37] C.C. Yu and C.C. Chang, "Distributed Query Processing," *ACM Computing Surveys*, vol. 16, no. 4, 1984.



member of the ACM and the IEEE Computer Society.

Kian-Lee Tan received the BSc (Hons) and PhD degrees in computer science from the National University of Singapore (NUS), in 1989 and 1994 respectively. He is currently an assistant professor in the Department of Computer Science, NUS. His major research interests include multimedia information retrieval, wireless information retrieval, query processing and optimization in multiprocessor and distributed systems, and database performance. He is a



Beng Chin Ooi received the BSc and PhD degree in computer science from Monash University, Australia, in 1985 and 1989, respectively. He was with the Institute of Systems Science from 1989 to 1991 before joining the Department of Computer Science, National University of Singapore. His research interests include database performance issues, multimedia databases and applications, high-dimensional databases, and GIS. He is the author of a

monograph entitled *Efficient Query Processing in Geographic Information Systems*, (Springer-Verlag, LCNS #471, 1990), and a coauthor of a monograph entitled *Indexing Techniques for Advanced Database Applications*, (Kluwer Academic Publishers, 1997). He has published more than 50 conference/journal papers and served as a PC member for a number of international conferences (including ACM SIGMOD, VLDB, EDBT) and serves as an editor for *Geoinformatics*, *International Journal of GIS* and the *Journal on Universal Computer Science*. He is a member of the ACM and the IEEE Computer Society.



David J. Abel is a chief research scientist with CSIRO Mathematical and Information Sciences. After a period in operations research (PhD, 1978), he worked on spatial access methods, spatial data models, multidatabase, and spatial decision support systems until 1995. His current specialization is interoperability in large-scale networks with high heterogeneity.