Query Optimization for a Distributed Geographic Information System

Fangju Wang

Abstract
Distributed geographic information systems (GISs) have advantages in data sharing, reliability, efficiency, and system growth. Query optimization substantially affects the performance of a distributed GIS. In developing a system, query optimization is one of the technical issues that must first be addressed. A distributed GIS is different from a non-spatial distributed database and requires special techniques for query optimization.

In this paper, a set of query optimization techniques are presented that were developed in building a distributed GIS. Two new definitions of spatial operations are introduced that enable us to apply the well-developed operation-ordering approach for strategy generation. A Petri net-based strategy-modeling method is described that is aimed at facilitating strategy generation and cost estimation. A query optimization algorithm is presented. Cost functions and selectivity functions for spatial operations are described as well.

Introduction

Distributed Geographic Information Systems

In recent years, *distributed* GISs have attracted increasing interest. A distributed GIS is a collection of sites connected via a data communication network. Each site is an autonomous GIS that maintains data and processing functions. A distributed GIS provides transparent access to data stored at any of the sites. It presents a single database image and hides data distribution and connection paths. To the user, all the data and functions can be accessed as if they are provided at the local site.

Compared with isolated/centralized GISs, distributed GISs have many advantages. The most obvious advantage is the support for data sharing. In many situations, particularly with large data processing projects, data sharing dramatically improves productivity and reduces costs. Additional advantages include improved efficiency, higher reliability, and easier system growth. A distributed GIS can reduce response time. By distributing data properly, the time required for data transmission is minimized. Short response time is also achieved by distributing costly operations to multiple sites for parallel processing. Higher reliability is achieved by duplicating crucial data and functions at multiple sites. In a well-planned system, new computers are easily "plugged in" to incorporate more power. In a word, integrated with data communication networks, GISs may become more accessible, available, and powerful.

The advantages and importance of distributed GISs have been realized by GIS researchers and producers (McGregor, 1988; NCGIA, 1989; Meredith, 1995). Some organizational and institutional issues in developing distributed GISs, including the incentives and the impediments, have been addressed by Pinto and Onsrud (1995). Research has been conducted for developing distributed GISs, for example, by Edmondson (1992), Bernath (1992), Laurini (1993), and Goodman (1994). Recent work includes the DGIS project in Australia (DHPC Project Team, 1996), the DISGIS project in Norway (Norwegian Mapping Authority, 1997), and the geodata modeling technique for distributed GISs at Berkeley (Gardels, 1997). To facilitate geographic data sharing and interoperability, international and national standards have been developed, including the Open Geodata Interoperability Specification (OGIS) (Buehler and McKee, 1996), and the Spatial Archive and Interchange Format (SAIF) (British Columbia Survey and Resource Mapping Branch, 1994). Since 1995, web server-based systems have been developed for geographic data sharing, for example, the Alexandria Digital Library (ADL) (Smith *et al.*, 1996) and many commercial and non-commercial systems (Plewe, 1997). Most of these types of systems support "map-based" queries where a query is used to retrieve geographic data (usually the whole or part of a map) stored at a single remote site. However, progress is slow in building systems that support queries that request map features, evaluate spatial relationships, and involve maps stored at multiple sites. The slow progress may be due, partly, to the special technical problems that must be solved in developing distributed GISs, such as query optimization.

Query Optimization

Query optimization is the generation of efficient execution *strategies* for queries. Modern information systems, including advanced GISs, use non-procedural languages to express queries. For a non-procedural query, the system must generate a procedure of operations to execute it. Such a procedure is called a *strategy*. In a distributed system, the strategy determines the sites and order for executing operations, as well as the procedure for transmitting the requested data.

Several strategies may exist for a query, for example, a query requesting data about the regions that have a land cover of "bare soil" and a slope less than five degrees. If the land-cover map and the slope map are stored at two different sites and the query is originated at a third site, we may use at least the following two strategies to obtain the result. The first strategy is to transmit the two maps to the originating site, overlay them and select the result there. The second strategy is to select the regions with the specified cover type or slope at the sites where the maps are stored, transmit the result of one site to the other, overlay the intermediate results and transmit the final result to the originating site. In most situations, the second

Department of Computing and Information Science, University of Guelph, Guelph, Ontario N1G 2W1, Canada (fjwang@snowhite.cis.uoguelph.ca).

Photogrammetric Engineering & Remote Sensing
Vol. 65, No. 12, December 1999, pp. 1427–1438.

0099-1112/99/6512–1427$3.00/0

strategy is more efficient because it involves transmitting and processing less data.

The performance of two strategies may differ by several orders of magnitude, and the use of different strategies substantially affects system performance. In developing a distributed GIS for practical use, query optimization is among the first technical issues to be addressed.

To date, research on distributed GIS query optimization is limited. The most recently published work includes query optimization for the Alberta Land Related Information System (Igras, 1994) and the spatial join strategies for distributed GISs (Abel *et al.*, 1995). However, extensive research has been conducted on query optimization for ordinary (non-spatial) distributed databases, and a rich set of techniques have been developed (Yu and Chang, 1984; Ozsu and Valduriez, 1991). Previous research on query optimization for isolated/centralized GISs has mainly focused on spatial indexing (Ooi, 1990; Laurini and Thompson, 1992; Brinkhoff *et al.*, 1993; Leslie *et al.*, 1995; Nabil and Gangopadhyay, 1997). Many techniques from these two related fields (especially those for ordinary distributed databases) can be used for distributed GISs.

This paper relates a set of query optimization techniques for distributed GISs. First, the key issues in query optimization for a distributed GIS are identified. Two new definitions of spatial operations are introduced that enable us to apply the existing optimization techniques to a distributed GIS. Second, an optimization algorithm is presented, which includes a strategy generation procedure, a correctness analysis method, and a cost model. Cost and selectivity functions of spatial operations are described as well.

A Review and Identification of the Key Issues

Query Optimization Techniques for Distributed DBs

Most of the existing query optimization algorithms were developed for *relational* databases. Representative algorithms include those for Distributed INGRESS (Epstein *et al.*, 1978), R* (Lohman *et al.*, 1985), and the algorithms by Apers *et al.* (1983). The algorithms share the same major steps: *query decomposition* that decomposes a query into subqueries, each of which can be executed at a site; *data localization* that determines the data involved in each subquery; *global optimization* that generates a strategy for executing the subqueries; and *local optimization* in which individual subqueries are optimized. Of the four steps, the first two are relatively straightforward, and local optimization can be performed by using the techniques for isolated/centralized systems. Global optimization is the core step and is the most complicated. It consists of the tasks of *strategy generation* and *cost estimation*.

The above algorithms were developed to optimize *join queries* (i.e., queries in which the main operations are sequences of joins). The major approaches for strategy generation are *operation ordering* and *semijoin*. For a query, operation ordering involves generating a strategy from permutations of its operations. This approach is based on the facts that joins are commutative, and that joins coupled with some other operations are commutative. Semijoin reduces costs by transmitting subsets of the relations to be joined instead of the whole relations. Figure 1 illustrates the five steps in a semijoin procedure, in which the symbol Π denotes a relational projection operation. When using semijoin to join relations R_1 and R_2 that are stored at sites S_1 and S_2, respectively, projection is conducted on R_1 to select the attribute(s) to be compared, the projected attribute values (a vertical subset of R_1 that is denoted as R_3) are transmitted to S_2 and joined with R_2, and then the join result (a horizontal subset of R_2 that is denoted as R_4) is transmitted to S_1 to join with R_1. The final result is R_5.

To select the best from alternative strategies, a *cost model* is needed to estimate total costs or response time. A cost model

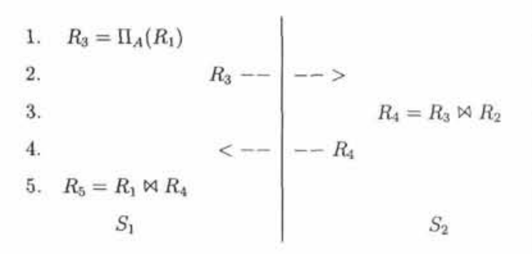


Figure 1. The five steps in semijoin.

may have the components of CPU costs, disk I/O costs, and transmission costs. To simplify cost estimation, most distributed non-spatial database systems (distributed DBs) designed for wide area networks ignore the local processing costs (Ozsu and Valduriez, 1991).

Identification of the Key Issues

A comparison between a distributed DB and a distributed GIS may help us identify and address the key issues in query optimization for a distributed GIS. It is assumed that the GIS to be compared stores vector data and that both systems are based on a relational model. The relational model is currently the most widely used database model in operational GISs. It is also a basis of the "object-relational" model proposed in the forthcoming standards of SQL:1999 (ISO/IEC, 1999a) and SQL/MM (ISO/IEC, 1999b). A digital map in a vector GIS is created as a collection of *spatial features*. The primary spatial features are points, lines, and polygons (Buehler and McKee, 1996). In a relational (or object-relational) GIS, a digital map is logically represented by the graphical display of its spatial features and a relation (table). Each tuple in the relation depicts a spatial feature (Aronoff, 1989; Laurini and Thompson, 1992).

A distributed DB and a distributed GIS have the common properties that data are distributed, queries can be decomposed into primary operations, and primary operations can be executed at different sites. The common properties suggest that query optimization in a distributed GIS can be performed in the similar steps (i.e., query decomposition, data localization, global optimization and local optimization).

However, a distributed DB and a distributed GIS are different in many aspects. The major differences that may affect query optimization, are with database operations:

- In a distributed DB, primary operations are relational and set operations. Relational and set operations are inadequate for GIS queries (Egenhofer, 1992). In a distributed GIS, a number of spatial operations are needed for overlay, buffering, and evaluating spatial relationships like adjacency, connection, and overlap, in addition to the above operations.
- The idea of semijoin cannot always reduce costs when applied with spatial operations. When a spatial operation is conducted on two maps, the "vertical subset" of a map is its spatial data. Spatial data of a digital map usually account for a major proportion of the data volume.
- The major operations in a distributed DB (i.e., join, selection, and projection) are commutative. Spatial operations do not show explicitly the properties of commutativity.
- Operations in a distributed DB do not create new attributes. They construct new relations by combining and re-arranging attributes of input relations. However, some spatial operations create new spatial features and attributes. For example, a buffering operation generates buffer zones. The zones are new spatial features and their properties have to be described by new attributes.
- Complexities of spatial operations vary widely. Many spatial operations have very costly procedures. For example, in overlaying two polygon maps, intersections of polygon boundary

lines must be detected. Computing time for the intersection detection may be proportional to $N_1 \cdot N_2 \cdot L_1 \cdot L_2$ where N_1 and N_2 are the numbers of polygon boundary lines on the two maps, and L_1 and L_2 are the average line lengths. Algorithms have been developed to improve efficiency, for example, by White (1978) and Franklin *et al.* (1989). However, intersection detection is still among the most time-consuming operations.

- A query optimizer needs to estimate the size of the result computed by an operation on the given input. The output of one operation can be the input to another operation. The cost of the latter depends on the size of its input. The size of an operation's output can be estimated using the operation's selectivity factor, which is the ratio of the size of its output to the size of its input. In a GIS, each spatial operation has its selectivity factor.

The differences suggest that in distributed GISs global optimization and local optimization, which decide how operations are performed on data, have to be conducted in special ways. While some techniques for isolated/centralized GISs can be applied for local optimization, special techniques are required for global optimization. The special techniques are the major task in developing query optimization techniques for distributed GISs.

Before presenting the algorithm and new techniques, we first identify the key issues in the development of them:

- Commutativity is the basis of operation ordering. To use operation ordering, we have to explore commutativity of the spatial as well as non-spatial operations in GIS queries.
- In developing a model for cost estimation, the following should be taken into consideration:
  - Local processing costs cannot be ignored. The costs of some spatial operations are comparable to the transmission costs on wide area networks.
  - Because complexities of spatial operations vary greatly, the cost of each operation should be estimated individually. This requires a cost function for each spatial operation.
  - To estimate sizes for intermediate results, a selectivity function should be defined for each spatial operation.
  - The cost model should be able to deal with parallel processing.

Exploring Commutativity of Spatial Operations

Operations in GIS Queries

Operations in GIS queries can be classified into non-spatial and spatial operations. A query may have operations from both classes. A *non-spatial operation* is conducted on non-spatial data only, (i.e., attribute data of digital maps or non-spatial relations). In a relational GIS, these operations include join (θ- or natural join), selection, and projection. Operations in this class are basically the same as those in non-spatial databases.

Spatial operations are performed on spatial features and their attribute data, and spatial operations produce new maps (Tomlin, 1990). In this research, we further classify spatial operations into two groups. The first group are operations that evaluate spatial relationships without creating new features on the output maps, these are called *spatial evaluations*. These include operations for evaluating adjacency, intersection, connection, overlap, and so on. The second group are operations that manipulate existing spatial features to create new features on the output maps; these are called *spatial manipulations*. These include polygon overlay, buffering, viewshed mapping, and so on.

In brief, we mainly deal with join, selection, spatial evaluations, and spatial manipulations in GIS queries. *The key to use operation ordering is to explore their commutativity.* In the following, we introduce two new definitions of spatial operations. Based on them, many spatial operations can be commuted with each other and commuted with the non-spatial operations.

General Definitions of Spatial Operations

Before introducing the definitions, it is necessary to describe the data structure of digital maps on which the spatial operations are conducted. The structure is similar to those applied in many existing systems: A digital map essentially consists of two components—a spatial data structure and a relation. The two components are stored and transmitted together.

In a "pure" relational GIS, the spatial data structure is usually a separate structure, for example, in the topological model (Aronoff, 1989). In an object-relational GIS, the spatial data structure can be constructed out of the spatial *abstract data types* defined in SQL/MM (ISO/IEC, 1999b). In either of the two types of structures, spatial data of a map can be viewed as a collection of spatial features, denoted as

$$\{f\} \tag{1}$$

where f is the generic representation of spatial features, and the bracket pair "{ }" denotes a set (or collection). Each spatial feature has a *feature identifier.*

The relation contains non-spatial attribute data, associates the spatial features with their attribute data, and specifies some integrity constraints. The *scheme* of a map relation can be

$$(A_0, A_1, \cdots, A_n) \tag{2}$$

where A_0 is the feature identifier, and $A_1, \cdots, A_n$ are other attributes ($n \geq 1$), which may also be feature identifiers. In this research, we use feature identifiers in this group to represent spatial relationships. This method will be discussed in the next section. A *relation* is a collection of tuples

$$\{t\} \tag{3}$$

where t is the generic representation of relational tuples defined by Expression 2. The spatial operations are defined in terms of input and output maps.

Spatial Evaluations

Spatial evaluations are binary in terms of the number of operands. On the spatial side, the input of a spatial evaluation is the spatial features of the two input maps and the output is the spatial features of a composite map. The composite map includes pairs of spatial features that satisfy the spatial relationship evaluated. The features in a pair are from the two input maps. For example, when we have a point map and a polygon map and we are evaluating the relationship of "point-within-polygon," if a point is within a polygon, the pair is included on the composite map. Formally, spatial evaluation can be expressed as

$$E(\{f\}_1, \{f\}_2) = \{(f_1, f_2)\} \tag{4}$$

where E is a spatial evaluation, $\{f\}_1$ and $\{f\}_2$ are the spatial feature collections of the two input maps, (f_1, f_2) is a pair of spatial features, and $f_1 \in \{f\}_2$ and $f_2 \in \{f\}_2$ satisfy the spatial relationship evaluated.

On the non-spatial side, the input is the relations of the input maps and the output is a composite relation. A tuple in the composite relation depicts a feature pair in Equation 4. A tuple is formed by concatenating the two tuples that depict the features in the pair. In the above example, a tuple in the composite relation indicates that a point and a polygon satisfy the relationship of "point-within-polygon." On this side, spatial evaluation E can be expressed as

$$E(\{t\}_1, \{t\}_2) = \{(t_1, t_2)\} \tag{5}$$

where $\{t\}_1$ and $\{t\}_2$ are the relations of the two input maps, (t_1, t_2)

is a tuple in the composite relation that is a concatenation of t_1 and t_2, and $t_1 \in \{t\}_1$ and $t_2 \in \{t\}_2$ depict two spatial features that satisfy the spatial relationship evaluated. The scheme of the composite relation is

$$(A_{1,0}, A_{1,1}, \cdots, A_{1,n_1}, A_{2,0}, A_{2,1}, \cdots, A_{2,n_2}) \tag{6}$$

where $A_{i,j}$ ($j = 0, 1, \ldots, n_i$) are attributes of the ith input relation ($i = 1, 2$). The values of $A_{1,0}$ and $A_{2,0}$ in a tuple represent the information that two features satisfy the spatial relationship evaluated.

Spatial Manipulations
A spatial manipulation has one or more input maps. On the spatial side, the input is the set(s) of spatial features of the input map(s) and the output is the spatial features of a composite map. The composite map includes the new spatial features created by the manipulation, as well as their "parent" features. An example can be found when we conduct a buffering operation on a map. The composite map includes buffer zones and also the features that are buffered. This group of operations can be formally expressed as

$$M(\{f\}_1, \ldots, \{f\}_m) = \{(f_c, f_1, \ldots, f_m)\} \tag{7}$$

where M is a spatial manipulation, $\{f\}_i$ is the set of spatial features of the ith input map ($i = 1, \ldots, m$), $(f_c, f_1, \ldots, f_m)$ is a group of spatial features on the composite map, f_c is a new feature created by the operation, $f_1 \in \{f\}_1, \ldots, f_m \in \{f\}_m$ are the parent features of f_c, and m is the number of input maps.

On the non-spatial side, the input is the relation of the input map(s) and the output is a composite relation. Each tuple in the output includes attributes of a new spatial feature and also attributes of its parent feature (or features). In the example of buffering, each tuple in the composite relation includes attributes of a buffer zone and attributes of the feature buffered by the zone. Formally, the operation can be expressed as

$$M(\{t\}_1, \ldots, \{t\}_m) = \{(t_c, t_1, \ldots, t_m)\} \tag{8}$$

where $\{t\}_i$ is the relation of the ith input map ($i = 1, \ldots, m$), $(t_c, t_1, \ldots, t_m)$ is a tuple in the composite relation formed by concatenation, t_c contains the attributes of a new feature created by the operation, and $t_1 \in \{t\}_1, \ldots, t_m \in \{t\}_m$ are the tuples that depict the parent features of f_c. The scheme of the composite relation is formally defined as

$$(A_{c,0}, A_{c,1}, \cdots, A_{c,n_c}, A_{1,0}, A_{1,1}, \cdots, A_{1,n_1}, \cdots, A_{m,0}, A_{m,1}, \cdots, A_{m,n_m}) \tag{9}$$

where $A_{c,j}$ ($j = 0, 1, \ldots, n_c$) are attributes of the newly created features and $A_{i,j}$ ($j = 0, 1, \ldots, n_i$) are attributes of the ith input relation ($1 \le i \le m$).

Commutativity of the Operations
A comparison between a type of spatial operations and relational θ-join may help us understand the commutativity of the former. Formally, relational θ-join can be expressed as

$$\bowtie_\theta (\{t\}_1, \{t\}_2) = \{(t_1, t_2)\} \tag{10}$$

where $\bowtie_\theta$ is θ-join, $\{t\}_i$ is the ith input relation ($i = 1, 2$), and (t_1, t_2) is a tuple in the joined relation formed by concatenating $t_1 \in \{t\}_1$ and $t_2 \in \{t\}_2$ that satisfy θ. The scheme of the joined relation is

$$(A_{1,1}, \cdots, A_{1,n_1}, A_{2,1}, \cdots, A_{2,n_2}) \tag{11}$$

where $A_{i,j}$ ($j = 1, \ldots, n_i$) are attributes of the ith input relation ($i = 1, 2$). The commutativity properties of θ-join can be found in Ozsu and Valduriez (1991) and Silberschatz *et al.* (1997).

By comparing Equations 4, 5, and 6 with Equations 10 and 11, we can observe that the spatial evaluations are similar to θ-join in terms of the relationship between the input and output: the output is a collection of pairs that include the input objects that satisfy a condition.

By comparing Equations 7, 8, and 9 with Equations 10 and 11, we can observe that, when the newly created features are not considered, spatial manipulations are similar to θ-join in terms of the relationship between the input and output: the output is a collection of groups that include the input objects that satisfy a condition. Because of the similarity, the spatial evaluations and spatial manipulations thus defined have the commutativity properties similar to those of θ-join. In the following, we list some of the properties that are the most useful in strategy generation. A more formal description of the properties and the proof of them can be obtained from the author.

Spatial evaluations have the following properties:

- Two spatial evaluations can be commuted,
- A spatial evaluation and a join can be commuted, and
- A spatial evaluation and a selection can be commuted.

Spatial manipulations have the following properties when the newly created features are not considered:

- A spatial manipulation and a spatial evaluation can be commuted,
- A spatial manipulation and a join can be commuted, and
- A spatial manipulation and a selection can be commuted.

The two definitions provide a conceptual framework for query optimization in a distributed GIS. Spatial operations implemented on the definitions have the properties of commutativity. The properties allow us to apply the operation ordering technique. We may order some operations of a query in a way such that the query can be executed efficiently. Note that, when a spatial evaluation, a join, or a selection is conducted on the new features created by a spatial manipulation, we have to execute the spatial manipulation first, and then other operations. This order cannot be reversed; otherwise, a strategy would be invalid. In strategy generation, it is an important task to avoid invalid strategies. The classification of spatial operations into evaluations and manipulations may ease this task considerably.

A Query Optimization Algorithm

An Overview of the Algorithm
The algorithm comprises three major steps:

(1) Query decomposition,
(2) Strategy generation, and
(3) Cost estimation.

The objective is to minimize response time. It is achieved by minimizing the sizes of the data to be processed and transmitted, and distributing costly spatial operations for parallel processing.

Operation ordering is used for strategy generation in this algorithm. Differing from Distributed INGRESS (Epstein *et al.*, 1978) and R* (Lohman *et al.*, 1985), which are based on operation ordering, this algorithm conducts optimization at compilation time and is not based on the exhaustive search of the solution space. The algorithm is aimed at generating a good strategy instead of the best. To reduce the number of alternative strategies, a set of rules are used. Strategies are represented as directed graphs. Petri nets (Peterson, 1981) are used to model strategies for correctness analysis and cost estimation. A set of cost functions and selectivity functions are defined for cost estimation.

In describing the algorithm, we use an extended SQL:1999 to express queries. A query in the language is of the form "**select-from-where**." The relations in the **from** clause may be digital maps or non-spatial relations. To express spatial operations, two groups of functions are defined in the language. The first group include functions "Overlay" and "Buffer" for spatial manipulations. The second group includes functions for conducting spatial evaluations. In the following discussion, the second group is called *spatial predicates*. The functions in the first group are used in **from** clauses and those in the second group used in **where** clauses. More details about the language can be obtained by contacting the author.

Query Decomposition

Query decomposition is an indispensable step in query optimization for non-spatial and spatial databases (Özsu and Valduriez, 1991; Ooi, 1990). In query decomposition, a query is decomposed into *subqueries*. A subquery here refers to an ordinary query that is decomposed from a user query and can be executed independently at a site. It is different from a subquery defined in the SQL standards. Query decomposition is conducted in four steps.

Step 1: Query Normalization

The search condition in the **where** clause is normalized into conjunctive normal form

$$C_1 \textbf{ and } C_2 \textbf{ and } \cdots \textbf{ and } C_n$$

where C_i $(1 \le i \le n)$ is the ith *conjunctive* term that is a disjunctive composition of *disjunctive terms*:

$$C_i = D_{i1} \textbf{ or } D_{i2} \textbf{ or } \cdots \textbf{ or } D_{ip}$$

where $p \ge 1$. A disjunctive term may be a spatial predicate (P), a "variable-operator-constant" (VC) predicate, or a "variable-operator-variable" (VV) predicate. "P" represents a spatial evaluation, "VC" represents a selection operation, and "VV" represents a join operation.

Step 2: Creating a Collecting Subquery

In a distributed GIS, a subquery is needed at the originating site for collecting intermediate results. It forms and displays the final result. This subquery is called a *collecting subquery*. Its target list is the same as the original query and its **where** clause is empty. The collecting subquery is created by splitting the original query

select $A_1, A_2, \ldots, A_T$
from $\{R\}$
where C

into a collecting subquery

select $A_1, A_2, \ldots, A_T$
from $\{R\}'$

and a subquery

select *
from $\{R\}$
where C

where $A_1, A_2, \ldots, A_T$ is the target list, $\{R\}$ denotes a set of relations (digital maps or non-spatial relations), and $\{R\}' \subseteq \{R\}$ is the relations involved in the target list. In the rest of the paper, we use $\{R\}$ to denote a set of relations, and we may express a condition as $C(R_1, \cdots, R_c)$ to indicate that $R_1, \cdots, R_c$ are relations involved in the condition.

Step 3: Separating Subqueries Containing Spatial Manipulations

If the non-collecting subquery has a spatial manipulation function in its **from** clause, a subquery is created to contain the function. Such a subquery is termed a *manipulating subquery*. Formally, this step decomposes query

select *
from $M(R_1, \ldots, R_m)$ as $R_{m+1}, R_{m+2}, \ldots, R_n$
where C

into a manipulating subquery

select *
from $M(R_1, \ldots, R_m)$ as R_{m+1}

and a subquery

select *
from $R_{m+1}, R_{m+2}, \ldots, R_n$
where C

where M is a spatial manipulation function; $R_1, \ldots, R_m$ are input of the function $(1 \le m \le 2)$; R_{m+1} is the derived relation; and $R_{m+2}, \cdots, R_n$ are other relations. Note that the second resultant subquery has the derived relation but not the function in its **from** clause.

Step 4: Decomposing the Non-Manipulating Subquery

In this step, the non-manipulating subquery is decomposed into subqueries each of which has one conjunctive term in its **where** clause. Formally, the subquery constructed using the ith conjunctive term, denoted by $C_i(\{R\})$, is

select *
from $\{R\}$
where $C_i(\{R\})$.

A Sample Query

The following is a sample query, and the subqueries that are decomposed from it. This query includes a spatial manipulation function and three spatial predicates. Its **where** clause has all the three types of terms (P, VC, and VV). This query can be used to display the regions that have a cover type of "bare soil" and a drainage class of "poor," and locate within parcels that are owned by the province. The parcels are not adjacent to Lake Ontario and have distances no less than 10 km to residential areas. In this query, the names in the upper case are relation names. In each "map" relation, the feature identifier is the attribute that has the same name as the relation but is in the lower case.

```
select REGION.region
from   Overlay(COVER,DRAINAGE) as REGION,
         OWNER, USE, LAKE, PARCEL
where  COVER.cover_type = 'bare soil' and
       DRAINAGE.class = 'poor' and
       PARCEL.o_id = OWNER.o_id and
       OWNER.owner_name = 'province' and
       not Adjacent(PARCEL.parcel,LAKE.lake) and
       LAKE.lake_name = 'Lake Ontario' and
       Distance(PARCEL.parcel,USE.use,'> = 10') and
       USE.use_type = 'residential' and
       Contain(PARCEL.parcel,REGION.region)
```

The sample query is decomposed into eleven subqueries. Subquery 0 is the collecting subquery and Subquery 1 is a manipulating subquery. Each of the rest contains a conjunctive term in the **where** clause of the original subquery.

```
q0. select REGION.region
    from  REGION

q1. select *
    from  Overlay(COVER,DRAINAGE) as REGION

q2. select  *
    from   COVER
    where COVER.cover__type = 'bare soil'

q3. select  *
    from   DRAINAGE
    where DRAINAGE.class = 'poor'

q4. select  *
    from   PARCEL, OWNER
    where PARCEL.o__id = OWNER.o__id

q5. select  *
    from   OWNER
    where OWNER.owner__name = 'province'

q6. select  *
    from   PARCEL, LAKE
    where not Adjacent(PARCEL.parcel,LAKE.lake)

q7. select  *
    from   LAKE
    where LAKE.lake__name = 'Lake Ontario'

q8. select  *
    from   PARCEL, USE
    where Distance(PARCEL.parcel,USE.use, '> = 10')

q9. select  *
    from   USE
    where USE.use__type = 'residential'

q10. select  *
     from   REGION, PARCEL
     where Contain(PARCEL.parcel,REGION.region)
```

Strategy Generation

Strategy generation has three steps. In the first step, query graphs are created. In a query graph, data transmissions are decided. In the second step, the query graphs are analyzed for eliminating the incorrect ones. In the third step, strategies are generated from the correct graphs.

Step 1: Generating Query Graphs

A query graph is a directed graph $G(Q, T)$ where Q is a set of nodes representing subqueries: $Q = \{q\}$, and T represents data transmissions between the subqueries. $t_{ij}^R \in T$ denotes the transmission of relation R from q_i to q_j. A query graph is actually a strategy without considering the sites to execute subqueries. By allocating subqueries to different sites, a query graph can be developed into an executable strategy.

In a query graph, the order for executing subqueries is represented by the direction of data transmissions. If q_i should be executed before q_j, the data transmission between them must be from q_i to q_j. In other words, in $G(Q, T)$, $t_{ij}^R \in T$ indicates that $q_i \in Q$ should be executed before $q_j \in Q$.

As discussed before, spatial evaluations can be commuted with each other and commuted with joins and selections, and spatial manipulations can be commuted with other operations when only the input maps are concerned. The properties of commutativity enable us to apply operation ordering to generate strategies. Of the four types of operations, spatial manipulations need special attention. A spatial manipulation creates new features and attributes. In a valid strategy, it must be executed before the new features and their attributes are processed by other operations.

To avoid exhaustive search and to prevent invalid placement of spatial manipulation functions, the following rules are used in an initial step of query graph generation:

(1) A selection should be conducted before a join, spatial evaluation, or spatial manipulation.
(2) A join should be conducted before a spatial manipulation, because the join condition may reduce the size of the input to the latter.
(3) A spatial evaluation should be conducted before a spatial manipulation, because the spatial relationship to be evaluated may reduce the size of the input to the latter.
(4) When the output of a spatial manipulation is in the input to a selection, join, spatial evaluation, or another spatial manipulation, the spatial manipulation must be conducted first.

The following is the procedure for query graph generation. The input is the subqueries decomposed from a user query. The output is a set of query graphs, denoted as $\{G\}$. In the following, $\{R\}_i$ denotes the relations in the **from** clause of subquery q_i:

```
/* Create nodes */
Create $Q = \{q\}$;
/* Add non-directed edges that represent data sharing */
Create $e_{ij}^R$ for each $R \in \{R\}_i \cap \{R\}_j$;
/* Convert the edges into directed edges that represent
   transmissions*/
For each $e_{ij}^R$
    If (($q_j$ is a collecting subquery) or
      ($R$ is the output of a spatial manipulation function in
        $q_i$)) Then Change $e_{ij}^R$ into $t_{ij}^R$,
    Else
        If (($R$ appears in a VC term in $q_i$) or
              ($R$ appears in a VV term in $q_i$ and in a spatial
                 manipulation function in $q_j$) or
              ($R$ appears in a P term in $q_i$ and in a spatial
                 manipulation function in $q_j$)) Then
                 Change $e_{ij}^R$ into $t_{ij}^R$,
        EndIf
    EndIf
EndFor
$\{G\} = G$;
For each $G \in \{G\}$ that has an $e_{ij}^R$ remaining
    Remove $G$ from $\{G\}$;
    Duplicate $G$ such that one copy has $t_{ij}^R$ and the other
      has $t_{ji}^R$;
    Add the two copies into $\{G\}$;
EndFor
/* Remove circles */
For each $G \in \{G\}$ with $q_i \in G$ such
  that there is an $R \in \{R\}_i$ that has $n > 1$ incoming edges
    Remove $G$ from $\{G\}$;
    Duplicate $G$ such that each copy has one incoming
      edge for $R$;
    Add the copies of $G$ into $\{G\}$;
EndFor
For each $G \in \{G\}$ with $q_i \in G$ such that there is an $R \in \{R\}_i$
  that has $n > 1$ outgoing edges
    Remove $G$ from $\{G\}$;
    Duplicate $G$ such that each copy has one outgoing edge
      for $R$;
```

```
        Add the copies of $G$ into $\{G\}$;
    EndFor
```

Figure 2 illustrates two query graphs that are generated out of the subqueries decomposed from the sample query. In the graphs, relation names are associated with edges to indicate the relations to be transmitted.

Step 2: Analyzing Graph Correctness Using Petri Nets
Of the query graphs generated in Step 1, some may not be able to produce correct results, for example, the one in Figure 2b. Such graphs must be discarded before developing strategies. The criterion for a correct query graph is that it must satisfy the two conditions:

- Let $\{R\}$ be the relations in the **from** clause of the original query. Each $R \in \{R\}$ must be sequentially transmitted to all the nodes that represent the subqueries that have R in their **from** clauses.
- Each node, except for the one modeling the collecting subquery, has at least one outgoing data transmission.

It can be observed that the graph in Figure 2a satisfies this criterion but the graph in Figure 2b does not.

Although directed graphs are useful methods for modeling queries (Silberschatz, 1997), they lack certain mechanisms required for query analysis. In this research, Petri nets (Peterson, 1981) were used to model strategies (that have been expressed as query graphs) for correctness analysis and later for cost estimation. Petri nets have well-developed methods for describing and analyzing the flow of information and control in systems, particularly systems like distributed query strategies that may exhibit asynchronous and concurrent activities. A Petri net is a directed graph. It has two sets of nodes: *places* and *transitions*. The nodes are connected by arcs from places to transitions or from transitions to places. If an arc is directed from node i to node j, i is said to be an input of j and j to be an output of i.

In a Petri net, there are *tokens* that may represent resources, data, and so on. The execution of a Petri net is controlled by the position and movement of tokens in the net. Tokens are initially assigned to some places. A *marking* of a Petri net is an assignment of tokens to some places. A marking μ is represented as a vector $\vec{\mu} = (\mu_1, \mu_2, \cdots, \mu_n)$ where μ_i $(1 \leq i \leq n)$ is the number of tokens in the ith place and n is the number of places in the net. Given a Petri net and a marking $\vec{\mu}$, *firing* a transition s_j produces a new marking $\vec{\mu}' = \delta(\vec{\mu}, s_j)$. A transition must be *enabled* in order to fire. A transition is enabled when each of its input places has a token. The transition fires by removing the enabling tokens from the input places and generating new tokens. The new tokens are deposited in the output places of the transition.

As described before, a query graph has nodes representing subqueries and directed edges representing planned data transmissions. In modeling a query graph, we use transitions to model subqueries and data transmissions, places to model input and output relations of subqueries and transmissions, and tokens to model data. To simulate the execution of a strategy, we use firings of transitions to model data processing and transmitting. To reduce analysis costs, we may use a single transition to model a subquery and the transmission that moves data from the subquery. A transition has one or more input places representing the relations in the **from** clause of the corresponding subquery and one or more output places representing the relations generated by and transmitted from the subquery. If there is a $t^R_{i,j}$ in the query graph, the output place of q_i labeled with R is merged with the input place of q_j labeled with R.

In a Petri net that models a strategy, one token is initially created for each base relation and assigned to a place that has no input transitions. Such an assignment is called an *initial marking*. Applying a conjunctive search condition term (VC, VV, or P) or a spatial manipulation function to a relation is modeled by passing the relation's token to the corresponding transition (to enable it) and firing the transition. If a transition models a search condition term, it moves the tokens in its input places to its output place when it fires. If a transition models a spatial manipulation, it moves the tokens to and creates a new token in its output place. The new token models the derived relation.

Figure 3 illustrates the graphical representation of two Petri nets with initial markings. The nets model the two query graphs in Figure 2. In the graphical representation, a circle ○ represents a place, a bar | represents a transition, and a bullet ● represents a token.

Correctness analysis of query graphs can be converted into the well-studied *reachability problem*. In Petri net N with marking $\vec{\mu}$, if a new marking $\vec{\mu}'$ can be produced by successive transition firings, $\vec{\mu}'$ is said to be *reachable* from $\vec{\mu}$. All the markings of N that are reachable from $\vec{\mu}$, form the *reachability set*, denoted as $\mathfrak{R}(N,\vec{\mu})$. The reachability problem can be stated as "Given an N with marking $\vec{\mu}$ and a marking $\vec{\mu}'$, is $\vec{\mu}' \in \mathfrak{R}(N, \vec{\mu})$?". An effective technique for the problem is the *reachability tree*. For a reachability problem, the tree root represents the initial marking, the nodes represent markings reachable from the initial, the leaves usually represent the "final" markings in which no more transitions can fire, and the edge directed from $\vec{\mu}'$ to $\vec{\mu}''$ represents the firing of a transition that changes $\vec{\mu}'$ into $\vec{\mu}''$. A path from the root to a leaf is a sequence that includes firings that change the initial marking to a final marking, and also includes the initial, final, as well as intermediate markings.

The criterion for a correct query graph can be translated into "given an initial marking, each transition must be able to fire and all the tokens must be able to arrive at the output of the transition that models the collecting subquery." Take the two nets in Figure 3 as examples. It can be observed that all the transitions in Figure 3a can fire and all the tokens can arrive the output of q_0. However, the transition in Figure 3b that represents q_4 may not fire because an input place of it will never have a token, and the token for OWNER cannot get to the output of q_0.

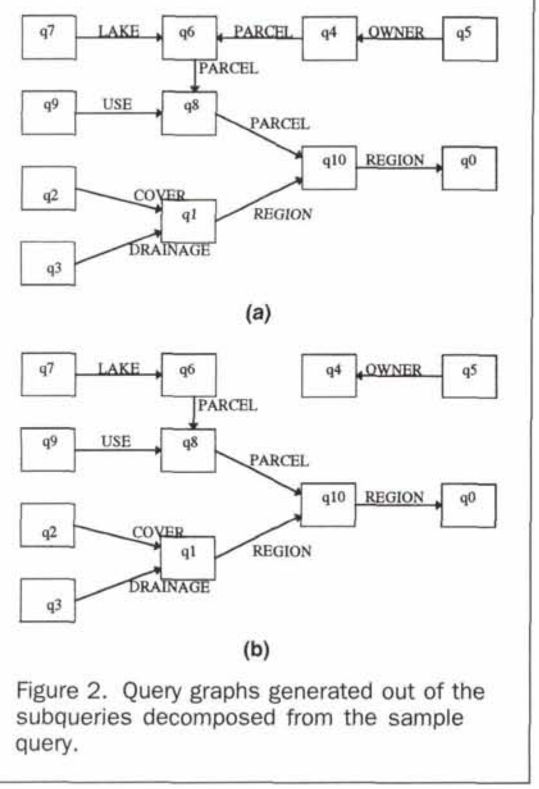


Figure 2. Query graphs generated out of the subqueries decomposed from the sample query.

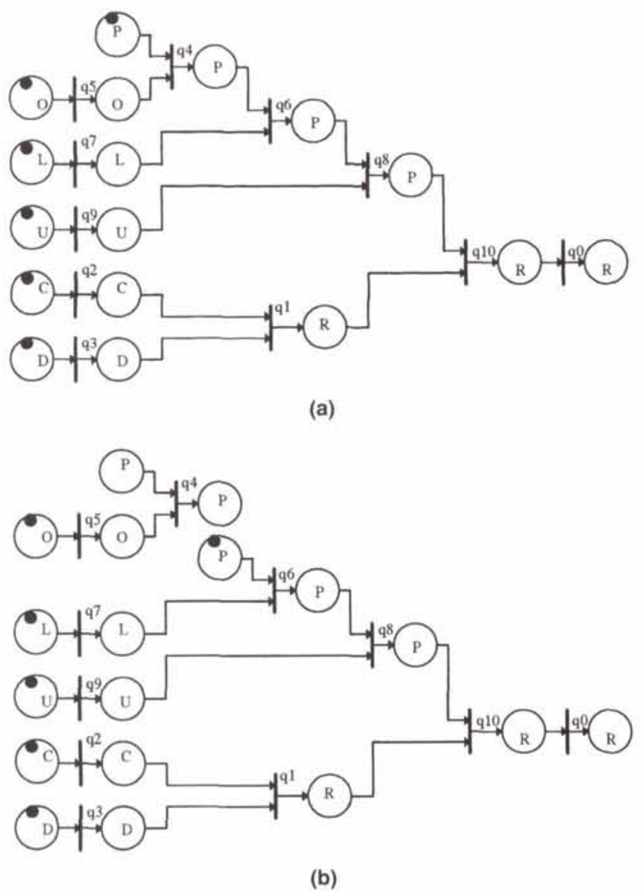


Figure 3. Petri nets modeling the query graphs in Figure 2 (C-COVER, D-DRAINAGE, R-REGION, O-OWNER, U-USE, L-LAKE, P-PARCEL.)

In a Petri net that models a query graph, any transition has no tokens in its output place until it fires. If a transition can fire, we will be able to find a reachable marking that indicates that there is a token in the output place. For each net we search the reachability tree for a path $\mathcal{P}$, such that for each transition s_i in the net, there exists a marking $\vec{\mu}'$ on $\mathcal{P}$ and $\mu_i \in \vec{\mu}'$ is not zero where μ_i denotes the number of tokens in the output place of s_i. If such a path can be found, the query graph modeled by the net is correct. A reachability tree may have more than one path. For a correct query graph, the paths lead to leaves representing the same marking. That is, the paths are equivalent in effect. The difference between two equivalent paths is that concurrent transitions have different orders in them. The creation of reachability trees will not be discussed here. It can be found in the literature on Petri nets, for example, Peterson (1981).

For the sample query, two correct query graphs are generated. One is illustrated in Figure 2a. The other one is $(\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9, q_{10}\}, \{t^{\mathrm{OWNER}}_{5,4}, t^{\mathrm{PARCEL}}_{4,8}, t^{\mathrm{PARCEL}}_{8,6}, t^{\mathrm{PARCEL}}_{6,10}, t^{\mathrm{LAKE}}_{7,6}, t^{\mathrm{COVER}}_{2,1}, t^{\mathrm{USE}}_{9,8}, t^{\mathrm{DRAINAGE}}_{3,1}, t^{\mathrm{REGION}}_{1,10}, t^{\mathrm{REGION}}_{10,0}\})$.

Step 3: Generating Alternative Strategies

Strategies are generated from the correct query graphs. A strategy is represented as a query graph $G(Q, T)$ with the q's in Q grouped. Each group is the subqueries allocated to the same site. The groups can be executed in parallel. The following is the procedure used for generating alternative strategies from a set of query graphs $\{G\}$.

```
For each $G(Q,T) \in \{G\}$
    While (There are $q$'s $\in Q$ to be allocated)
        For each $q_i \in Q$ to be allocated
            If ($\exists \{R\}' \subseteq \{R\}_i$ that do not have incoming edges)
                Then
                If (The $R$'s $\in \{R\}'$ are stored at the same site)
                    Then
                    Allocate $q_i$ to that site;
                Else
                    If (There is an $R \in \{R\}'$ that has a size much
                        larger than the rest) Then
                        Allocate $q_i$ to the site where $R$ is stored;
                    Else /* The $R$'s $\in \{R\}'$ have similar sizes */
                        Remove $G$ from $\{G\}$;
                        Duplicate $G$ such that each copy has $q_i$ allo-
                            cated to one of the sites there the $R$'s are
                            stored;
                        Add the copies of $G$ into $\{G\}$;
                    EndIf
                EndIf
            Else /* All the $R$'s $\in \{R\}'$; have incoming edges */
                If (The senders of all the $R$'s $\in \{R\}_i$ have been
                allocated) Then
                Remove $G$ from $\{G\}$;
                Duplicate $G$ such that each copy has $q_i$ allo-
                    cated to one of the sites there the senders
                    are allocated;
                Add the copies of $G$ into $\{G\}$;
                EndIf
            EndIf
        EndFor
    EndWhile
EndFor
```

Subqueries in each group are then ordered to schedule their execution by the local processor. The subqueries are ordered based on the rule that a receiver subquery can be executed only after the execution of all of its sender subqueries. Subquery ordering may be considered as an issue of local optimization. Because different orders of a group may affect the cost of a global strategy, we include it in the procedure of global strategy generation.

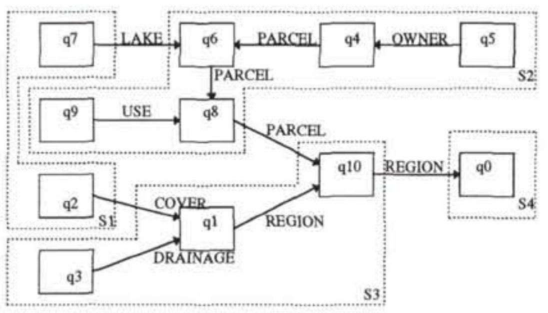
From the query graph in Figure 2a, four strategies are generated. The following is the data distribution and map sizes when the strategies are generated: S1 stores COVER (8.5M) and LAKE (1.7M); S2 stores PARCEL (3.2M), OWNER (400K), and USE (6.3M); and S3 stores DRAINAGE (4.4M). The query is originated at S4. Figure 4 shows a strategy generated with the subqueries in each group ordered. We use dash lines to indicate subquery allocation and S_i $(i = 1, 2, \ldots, 4)$ to represent different sites. The subquery assignments in the other three strategies are

Strategy 2: S1: q_1, q_2, q_7; S2: $q_4, q_5, q_6, q_8, q_9, q_{10}$; S3: q_3; S4: q_0.
Strategy 3: S1: q_1, q_2, q_7, q_{10}; S2: q_4, q_5, q_6, q_8, q_9; S3: q_3; S4: q_0.
Strategy 4: S1: q_2, q_7; S2: q_4, q_5, q_6, q_8, q_9; S3: q_1, q_3, q_{10}; S4: q_0.

Cost Estimation

The Cost Model

To select the best strategy from the alternatives, a cost model is needed. Although computational complexity of individual spatial operations has been well studied (Preparata and Shamos, 1988), so far much less work has been reported on cost modeling for GIS queries. In this research, a cost model has been developed to estimate response time. The time is estimated in terms of computing time of subqueries (including disk I/O time) and data transmission time. The model is based on the Petri nets generated in the correctness analysis.



S1: q7, q2
S2: q5, q4, q6, q9, q8
S3: q3, q1, q10
S4: q0

Figure 4. A strategy generated from the query graph in Figure 2a.

To estimate response time for a strategy (modeled as a Petri net), we first identify the starting transitions and ending transition. A starting transition does not have any preceding transition. An ending transition is the one modeling the collecting subquery.

In the cost model, each transition s has *firing time* τ_f^s and *ending time* τ_e^s, each place p has *enabling time* τ_a^p representing the time the data (represented by a token) are available, and each token k is associated with a size ν^k representing the size of the corresponding data.

The response time of a strategy is estimated as

$$\text{response time} = \tau_e^{s_e} \tag{12}$$

where s_e is the ending transition.

As mentioned before, a transition may model a subquery and a transmission. The ending time of transition s is

$$\tau_e^s = \tau_f^s + \tau_c^s + \tau_t^s \tag{13}$$

where τ_f^s is the firing time, τ_c^s is the computing time of the subquery, and τ_t^s is the time required for the transmission.

The firing time of transition s is estimated as

$$\tau_f^s = \max\left(\tau_e^{s'}, \max_{j=1}^{l}\left(\tau_a^{p_j}\right)\right) \tag{14}$$

where s' is the subquery that is executed immediately before s at the same site, p_j is an input place of s, and l is the number of input places of s.

The enabling time of place p is estimated as

$$\tau_a^p = \begin{cases} 0 & \text{if } p \text{ is initially asigned a token} \\ \tau_e^{s_i} & \text{if } p \text{ is the output of transition } s_i. \end{cases} \tag{15}$$

The computing time of a subquery is estimated by using the cost function(s) of the subquery's operation(s) and the sizes of the operands. A subquery may have one or more disjunctive terms (operations). We use the sum to approximate the computing time: i.e.,

$$\tau_c^s = \sum_i f_i \tag{16}$$

where f_i is the cost function of the ith operation. The cost functions will be discussed later.

Transmitting time can be estimated in terms of the network speed and the size of the data to be transmitted (Ceri and Pelagatti, 1984).

$$\tau_t^s = \begin{cases} 0 & \text{if the transmission is between two subqueries at the same site} \\ \tau_0^{i,j} + \tau_u^{i,j} \cdot \nu^k & \text{if the transmission is between sites } i \text{ and } j \end{cases} \tag{17}$$

where $\tau_0^{i,j}$ is the time required to initiating a transmission between sites i and j, $\tau_u^{i,j}$ is the unitary transmission time between the two sites, ank ν_k is the size of the data to be transmitted.

For a token that is initially assigned to a place, its size can be directly taken from database statistics. For a token that represents an intermediate result, its size is estimated based on database statistics and operation selectivity. The estimation is discussed in the next section.

Database Statistics and Size Estimation for Intermediate Results

We store statistics information about each digital map and each non-spatial relation in its metadata set. The statistics are calculated when the map or relation is loaded into the database or generated by an operation. The metadata set is attached with the map or relation when it is transmitted.

The statistics about a digital map are extracted from its spatial data structure and relation. The statistics from the spatial data structure include

- the feature *cardinality*, which is the number of features on the map;
- the average data size of the spatial features on the map;
- the density of the features, which is calculated as the number of features per unit area; and
- other information, including the average area of polygons, the average length of lines, the average number of polygon boundary lines, the average number of lines that a line may connect, the average number of polygons that a polygon may be adjacent to, and so on.

The statistics extracted from the relation include the minimum and maximum values and the number of distinct values for each attribute, the number of tuples, and the tuple size. A histogram is stored for each attribute that may be involved in (relational) selections.

The result of a spatial evaluation is a composite map and a relation. The size of the result is the sum of their sizes. The composite map contains feature pairs that satisfy the spatial relationship evaluated. In the following, we use E to represent the spatial relationship, $A_{1,0}$ and $A_{2,0}$ to denote the two operands, and $A'_{1,0}$ and $A'_{2,0}$ to denote the features in the result ($A'_{1,0} \subseteq A_{1,0}$ and $A'_{2,0} \subseteq A'_{2,0}$). In most situations, features denoted by $A'_{1,0}$ or $A'_{2,0}$ are input to other operations.

The size of one type of features in the result, for example, $A'_{1,0}$, can be estimated as

$$v(A'_{1,0}) = v_{\text{avg}}(A'_{1,0}) \cdot [card(A_{1,0}) \cdot S^E_{A_{1,0}}(A_{1,0}, A_{2,0})] \tag{18}$$

where $v_{\text{avg}}(A'_{1,0})$ is the average size of $A'_{1,0}$, which is equal to $v_{\text{avg}}(A_{1,0})$ and can be found in the database statistics, $card(A_{1,0})$ is the cardinality of $A_{1,0}$, and $S^E_{A_{1,0}}$ is the selectivity function of the spatial evaluation for E with respect to $A_{1,0}$. i.e.,

$$S^E_{A_{1,0}}(A_{1,0}, A_{2,0}) = card(A'_{1,0})/card(A_{1,0}) \tag{19}$$

where $card(A'_{1,0})$ is the cardinality of $A'_{1,0}$. $card(A'_{1,0})$ can be estimated from $dens(A_{1,0})$ and $card(A_{2,0})$ where $dens(A_{1,0})$ is the density of $(A_{1,0})$ that can be found in the database statistics. For example, $card(A'_{1,0})$ in the result of applying Distance($A_{1,0}$, $A_{2,0}$, p) can be estimated as

$$card(A'_{1,0}) = buff(A_{2,0}) \cdot dens(A_{1,0}) \cdot card(A_{2,0}) \qquad (20)$$

where $buff(A_{2,0})$ is the area of some kind of buffers of $A_{2,0}$. The buffers are defined based on the nature of $A_{2,0}$ and p. For example, when $A_{2,0}$ represents points and p is "$\leq 20\ km$," the buffers are circles of 20 km radii.

The size of the relation can be estimated as the product of its tuple size and the cardinality estimated using Equation 20. The tuple size can be calculated from the tuple sizes of $A_{1,0}$ and $A_{2,0}$.

When the features created by a spatial manipulation are input to another operation, the size of the features should be estimated when estimating the cost of the latter. For results of spatial manipulations, the size estimation is more straightforward. The size of the composite map of an overlay operation can be estimated as the sum of the two input maps. The size of the composite map of a buffering operation can be estimated as the product of the cardinality of the input map and the average buffer size. The average data size of the buffers can be estimated from the average size of the input features.

Cost Functions for the Spatial Operations

The cost function for a spatial operation consists of the cost for processing spatial features and the cost for processing non-spatial attribute data. We estimate the two costs in terms of time. The cost for processing spatial features is a sum of two components

$$\text{Cost}_c + \text{Cost}_{io} \qquad (21)$$

where Cost_c is a computing cost, and Cost_{io} is a disk I/O cost.

Cost_c can be estimated in terms of the time required for addition, multiplication, and logical comparison operations. For each of the three types of operations, the time can be estimated as the product of the number of the operations and the time required to conduct an operation. Therefore, for an operation on a collection of spatial features, the cost is expressed as

$$\text{Cost}_c = \alpha \cdot \tau_{\text{add}} + \beta \cdot \tau_{\text{mul}} + \gamma \cdot \tau_{\text{comp}} \qquad (22)$$

where τ_{add}, τ_{mul}, and τ_{comp} are the times required for executing an addition, multiplication, and logical comparison operation, respectively, and α, ZB, and γ are the numbers of the three operations required for processing the spatial data. τ_{add}, τ_{mul}, and τ_{comp} are machine-dependent, while α, β, and γ are algorithm-dependent and are functions of cardinalities and other statistics about the features. In the following, we take α in the cost function for spatial evaluation "Within" as an example to illustrate how we formulated expressions for the numbers.

The operation is expressed as Within($A_{1,0}$, $A_{2,0}$) in a query. $A_{2,0}$ represents polygons. When $A_{1,0}$ represents points, this operation evaluates the "point-within-polygon" relationship. For a given set of points, this function can be used to find all the polygons within which the points are located. To determine if a point is within a polygon, we use the "half-line" method that is based on the Jordan Curve Theorem (Preparata and Shamos, 1988). A half-line starts from this point and extends to beyond the extent (the minimum-bounding rectangle) of the polygon. The half-line and boundary lines of the polygon may intersect. If the number of intersections is odd, the point is within the polygon. To determine if the half-line intersects a boundary line, we check each segment (straight line) of the boundary line. We use algorithm Pavlidis' (1982) for the intersection test.

The number of addition operations for checking a boundary segment can be determined from the intersection algorithm. We use α_0 to denote it. The number for checking a boundary line can be estimated as

$$l_{\text{avg}} \cdot \alpha_0 \qquad (23)$$

where l_{avg} is the average length of the boundary lines. The number for applying the half-line method to a point-polygon pair can be estimated as

$$n_{\text{avg}} \cdot l_{\text{avg}} \cdot \alpha_0 \qquad (24)$$

where n_{avg} is the average number of boundary lines per polygon.

The number of addition operations for a point can be estimated as

$$n_{\text{contain}} \cdot n_{\text{avg}} \cdot l_{\text{avg}} \cdot \alpha_0 \qquad (25)$$

where n_{contain} is the average number of polygons whose minimum-bounding rectangles contain the same point. The total number of addition operations for all the points, i.e., the α in Equation 22, can be estimated as

$$\alpha = card(A_{1,0}) \cdot n_{\text{contain}} \cdot n_{\text{avg}} \cdot l_{\text{avg}} \cdot \alpha_0 \qquad (26)$$

where $card(A_{1,0})$ is the cardinality of the points.

We can estimate β and γ in the same way. For a spatial operation and a set of spatial data, once α, β, and γ have been estimated, the total time required for conducting the operation can be estimated by using Equation 22.

Cost_{io} is the cost for fetching some spatial features from disk files into the main memory. In a GIS, spatial features of a map are stored in one or more disk files based on a clustering strategy. A spatial index may be associated with the files to facilitate the search for the features that satisfy certain spatial conditions. Usually, before a spatial operation is conducted, only a subset of the features are fetched from the files. For example, before the "Within" operation is conducted on a set of points, only the polygons whose minimum-bounding rectangles contain the points are retrieved from the disk file(s). For a spatial operation and a map, the time required for disk data retrieval, i.e., Cost_{io}, is a function of the clustering strategy, the page size of the files, the indexing method, the retrieval type (containment or intersection), and characteristics of the map involved (Wang and Sun, 1997). The characteristics include the data size, the feature density, and so on.

The cost for processing the non-spatial attribute data is estimated in terms of the time for disk I/O. For each type of the features that are included in the output, for example $A_{1,0}$, we estimate the cost as

$$\tau_{io} \cdot \nu_{\tau}(A_{1,0}) / \nu_{\text{page}} \qquad (27)$$

where τ_{io} is the I/O time for one page, $\nu_{\tau}(A_{1,0})$ is the size of the relation for $A_{1,0}$, and ν_{page} is the page size.

Tests

The optimization algorithm has been implemented in an experimental system, which is built on a group of workstations connected with a local area network. The machines include IBM RISC/6000 workstations and DEC Alpha workstations. All the machines run UNIX operating systems. The system is written in C++ based on the Common Object Request Broker Architecture (CORBA) (Siegel, 1996). The system communication is implemented using the Remote Procedure Call (RPC) system of the TCP/IP (Transmission Control Protocol/Internet Protocol).

The algorithm was tested when the experimental system had 34 digital maps distributed at eight sites. The maps had different sizes and statistics. The tests were conducted when the machines had no other application workload and there was little other traffic on the network. Twenty-five queries were used for the test. For each query, the best strategy or strategies by the algorithm were executed. For comparison, the best strategy and a couple of "bad" strategies generated manually were also executed. For each strategy, the response time was measured.

The algorithm generated good (not necessarily the best) strategies for all the queries except two. For about 65 percent of the queries, the algorithm generated the same best strategies as did manual optimization.

Table 1 lists some test results obtained by executing four strategies for the sample query, where "s" denotes seconds and "ms" denotes milliseconds. The second and third strategies were generated by the algorithm. The first and fourth were manually generated. The five maps cover an area in southwestern Ontario. The map scales are 1:50,000. The data distribution and map sizes can be found in the section on Strategy Generation. The originating site is S4. The four nodes were run on the same type of IBM RISC/6000 workstations. Each has a processor rated at 32.2 MIPS and 11.7 MFLOPS. The response time was measured when the network had speeds between about 600K per second and 1M per second. To reduce the effects of the variations in network speed, each of the four strategies was executed several times. The response times listed in Table 1 are an average.

The first strategy had a bad distribution of subqueries. All the subqueries (operations) were executed sequentially on the originating node (S4). All the data needed (five maps and a table (OWNER)) were transmitted to S4 before being processed. It was determined that more than half of the total time was spent on transmitting the data. The second and third strategies used parallel processing and had fewer data transmissions. Both strategies had four inter-site data transmissions. The difference is that the second strategy involved transmitting a subset of COVER while the third involved transmitting a subset of DRAINAGE. The third strategy had a slightly shorter response time, possibly because the subset of COVER was bigger than the subset of DRAINAGE. This was the best strategy selected by the algorithm. The response time for the second and third strategies included the time for optimization, which is about 1.13 seconds. The fourth strategy was a bad strategy because the overlay operation (q_1) was conducted before the selection operations (q_2 and q_3). The long response time was due to the long processing time for overlaying the entire maps of COVER and DRAINAGE.

Concluding Remarks

We have discussed the query optimization techniques developed in building an experimental system. The core part is the definitions of spatial operations, the strategy modeling method based on Petri nets, the model and functions for cost estimation, and the optimization algorithm.

TABLE 1. FOUR DIFFERENT STRATEGIES AND RESPONSE TIME.

Execution sequences		Response time
S4: q_2, q_3, q_5, q_7, q_9,	q_4, q_1, q_6, q_8, q_{10}, q_0	27s 663ms
S1: q_2, q_7	S2: q_5, q_9, q_4, q_6, q_8	
S3: q_3, q_1, q_{10}	S4: q_0	8s 426ms
S1: q_2, q_7, q_1, q_{10}	S2: q_5, q_9, q_4, q_6, q_8	
S3: q_3	S4: q_0	7s 741ms
S1: q_7	S2:q_5, q_9, q_4, q_6, q_8	
S3: q_1, q_2, q_3, q_{10}	S4: q_0	42s 517ms

The definitions of spatial operations may help bridge the gap between distributed GISs and the query optimization techniques for distributed DBs. Spatial and non-spatial operations are quite different. The definitions can unify them so that spatial queries can be optimized in a way similar to that of non-spatial queries.

The major advantage of using Petri nets for strategy modeling is the facilitation of strategy generation and cost estimation. By moving correctness analysis out of query graph generation, the procedure for graph generation has been largely simplified because correctness analysis is a very complicated step to develop. Cost estimation based on Petri nets is straightforward and easy to understand. Another advantage is processing efficiency: the reachability trees of Petri nets are finite (Peterson, 1981) and have relatively small depths for most queries. In addition, using the same nets for both correctness analysis and cost estimation may reduce optimization costs.

Acknowledgment

This research was supported by research grants from the Natural Sciences and Engineering Research Council (NSERC) of Canada.

References

Abel, D.J., B.C. Ooi, K. Tan, R. Power, and J.X. Yu, 1995. Spatial Join Strategies in Distributed Spatial DBMS, *Fourth International Symposium, Advances in Spatial Databases*, Portland, Maine: Springer-Verlag, New York, N.Y., pp. 348–367.

Apers, P.M.G., A.R. Hevner, and S.B. Yao, 1983. Optimization Algorithms for Distributed Queries, *IEEE Transactions of Software Engineering*, 9(1):57–68.

Aronoff, S., 1989. *Geographic Information Systems: A Management Perspective*, WDL Publications, Ottawa, 294 p.

Bernath, T., 1992. Distributed GIS Visualization System, *Proceedings of GIS/LIS'92*, San Jose, California, 1:51–57.

Brinkhoff, T., H. Horn, H. Kriegel, and R. Schneider, 1993. A Storage and Access Architecture for Efficient Query Processing in Spatial Database Systems, *Third International Symposium, Advances in Spatial Databases*, Singapore, Springer-Verlag, New York, N.Y., 1:357–376.

British Columbia Survey and Resource Mapping Branch, 1994. *Spatial Archive and Interchange Format: Formal Definition Release 3.1*, Province of British Columbia.

Buehler, K., and L. McKee, 1998. *The OpenGIS Guide, Third Edition*, Open GIS Consortium, Inc., Wayland, Massachusetts, 103 p.

Ceri, S., and G. Pelagatti, 1984. *Distributed Databases Principles & Systems*, McGraw Hill Book Company, New York, N.Y., 393 p.

DHPC Project Team, 1996. *Distributed Geographic Information Systems Project Concepts Discussion Document*, **http://www.dhpc.adelaide.edu.au/projects/dgis**.

Edmondson, P.H., 1992. Managing the Distributed GIS Infrastructure—An Organizational Perspective, *Proceedings of GIS/LIS'92*, San Jose, California, 1:196–207.

Egenhofer, M.J., 1992. Why Not SQL!, *International Journal of Geographical Information Systems*, 6(2):71–85.

Epstein, R., M. Stonebraker, and E. Wong, 1978. Distributed Query Processing in a Relational Database System, *Proceedings of the ACM-SIGMOD International Conference on Management of Data*, Austin, Texas, pp. 169–180.

Franklin, W.R., C. Narayanaswami, M. Kanhanhalli, D. Sun, M. Zhou, and P.Y.F. Wu, 1989. Uniform Grids: A Technique for Intersection Detection on Serial and Parallel Machines, *Proceedings, Auto-Carto 9*, American Society for Photogrammetry and Remote Sensing, pp. 100–109.

Gardels, K., 1997. *A Comprehensive Data Model for Distributed, Heterogeneous Geographic Information*, **http://regis.berkeley.edu/gardels**.

Goodman, J.N., 1994. Alberta Land Related Information System, a Federated Database System Case Study, *URISA 1994 Annual Conference Proceedings*, Washington, D.C.: Urban and Regional Information Systems Association, 1:421–431.

Igras, E., 1994. A Framework for Query Processing in a Federated Databased System: A Case Study, *URISA 1994 Annual Conference Proceedings*, Washington D.C.: Urban and Regional Information Systems Association, 1:167–178.

ISO/IEC, 1999a. *Database Language SQL—Part 2, SQL Foundation* (Approval Version), The International Organization for Standardization, New York, N.Y., 810 p.

———, 1999b. *Information Technology—Database Languages—SQL Multimedia and Applications Packages—Part 3: Spatial* (Approval Version), The International Organization for Standardization, New York, N.Y., 342 p.

Laurini, R., 1993. Sharing Geographic Information in Distributed Databases, *Proceedings of the 16th Urban Data Management Symposium*, Vienna, Austria, pp. 26–41.

Laurini, R., and D. Thompson, 1992. *Fundamentals of Spatial Information Systems*, Academic Press Ltd., San Diego, 680 p.

Leslie, H., R. Jain, D. Birdsall, and H. Yaghmai, 1995. Efficient Search of Multidimensional B-Trees, *Proceedings of the 21th VLDB Conference*, Zurich, Switzerland, pp. 710–719.

Lohman, G.M., C. Mohan, L. Haas, D.J. Daniels, B. Lindsay, P. Selinger, and P Wilms, 1985. Query Processing in R*, *Query Processing in Database systems*, (W. Kim, D. Reiner, and D. Batory, editors), Springer-Verlag, New York, pp. 31–47.

McGregor, D., 1988. Geographic Information System Trends, *Proceedings of GIS/LIS'88*, San Antonio, Texas, 2:915–921.

Meredith, P.H., 1995. Distributed GIS: If Its Time Is Now, Why Is it Resisted?, *Sharing Geographic Information*, (H.J. Onsrud and G. Rushton, editors), Center for Urban Policy Research, New Brunswick, N.J., pp. 7–21.

Nabil, R.A., and A. Gangopadhyay, 1997. *Database Issues in Geographic Information Systems*, Kluwer Academic Publishers, Boston, 136 p.

NCGIA (National Center for Geographic Information and Analysis), 1989. The Research Plan of the National Center for Geographic Information and Analysis, *International Journal of Geographical Information Systems*, 3(2):117–136.

Norwegian Mapping Authority, 1997. *Project Summary*, Norwegian Mapping Authority, **http://www.statkart.no/disgis**.

Ooi, B.C., 1990. *Efficient Query Processing in Geographic Information Systems*, Springer-Verlag, Berlin, 208 p.

Ozsu, M.T., and P. Valduriez, 1991. *Principles of Distributed Database Systems*, Prentice Hall, Englewood Cliffs, New Jersey, 562 p.

Pavlidis, T., 1982. *Algorithms for Graphics and Image Processing*, Rockville, Maryland, Computer Science Press, 416 p.

Peterson, J.L., 1981. *Petri Net Theory and the Modeling of Systems*, Prentice Hall, Englewood Cliffs, New Jersey, 290 p.

Pinto, J.K., and H.J. Onsrud, 1995. Sharing Geographic Information Across Organizational Boundaries: A Research Framework, *Sharing Geographic Information*, (H.J. Onsrud and G. Rushton, editors), Center for Urban Policy Research, New Brunswick, N.J., pp. 44–64.

Plewe, B., 1997. *GIS Online: Information Retrieval, Mapping, and the Internet*, On Word Press, Santa Fe, New Mexico, 311 p.

Preparata, F.P., and M.I. Shamos, 1988. *Computational Geometry an Introduction*, Springer-Verlag Inc., New York, 398 p.

Siegel, J., 1996. *CORBA Fundamentals and Programming*, John Wiley & Sons, Inc., New York, 693 p.

Silberschatz, A., H.F. Korth, and S. Sudarshan, 1997. *Database System Concepts, Third Edition*, WCB/McGraw-Hill, Boston, Massachusetts, 821 p.

Smith, T.R., D. Andresen, L. Carver, R. Dolin, C. Fischer, J. Frew, M. Goodchild, O. Ibarra, R.B. Kemp, R. Kothuri, M. Larsgaard, B.S. Manjunath, D. Nebert, J. Simpson, A. Wells, T. Yang, and Q. Zheng, 1996. The Alexandria Digital Library: Overview and WWW Prototype, *IEEE Computer*, 29(5):54–60.

Tomlin, C.D., 1990. *Geographic Information Systems and Cartographic Modeling*, Prentice Hall, Englewood Cliffs, N.J., 249 p.

Wang, F., and Y. Sun, 1997. Spatial Object Clustering for an Object-Relational GIS, *Proceedings of GIS/LIS'97*, 28–30 October, Cincinnati, Ohio.

White, D., 1978. A Design for Polygon Overlay, *Harvard Papers on Geographic Information Systems*, (G. Dutton, editor), Vol. 6.

Yu, C.T., and C.C. Chang, 1984. Distributed Query Processing, *Computing Surveys*, 16(4):399–433.

(Received 02 January 1998; revised and accepted 03 November 1998; revised 15 January 1999)