The Geometry of Browsing

Richard Beigel[1] and Egemen Tanin[2]

[1] Lehigh University, Dept. of EE&CS, 19 Memorial Dr W Ste 2,
Bethlehem, PA 18015-3084, USA
beigel@eecs.lehigh.edu

[2] University of Maryland, Human–Computer Interaction Laboratory,
College Park, MD 20742-3251, USA
egemen@cs.umd.edu

Abstract. We present a geometric counting problem that arises in browsing and solve it in constant time per query using nonexhaustive tables. On the other hand, we prove that several closely related problems require exhaustive tables, no matter how much time we allow per query.

1 Introduction

In this paper we address some algorithmic problems that arise in connection with browsing a really large collection of datasets. The interface paradigms we present have been developed and user-tested as part of the visual data mining effort (cf. [14]) at the Human–Computer Interaction Laboratory (HCIL) in the University of Maryland. The target application is the EOSDIS collection of datasets in development by the U.S. National Aeronautics and Space Administration (NASA).

1.1 What is EOSDIS?

The NASA EOSDIS (Earth Observing System Data and Information System) project is attempting to provide online access to a rapidly growing archive of scientific data about the Earth's land, water, and air. Data is collected from satellites, ships, aircraft, and ground crews, and stored in designated archive centers. Scientists, teachers, and the general public are given access to this data via the Internet.

HCIL is developing user interfaces that use the "dynamic query" paradigm and the "query preview" paradigm to facilitate the browsing and retrieval of data from this very large archive. The background for interfaces to EOSDIS is discussed further on the web at [6].

1.2 What are Dynamic Queries?

Dynamic queries are an interface paradigm that allow the user to interactively control query parameters and generate a rapidly animated visual display of database search results [1, 2, 5, 7, 8, 10–13, 15, 16]. As users adjust sliders or buttons, results are updated nearly continuously on the display. Each adjustment to a slider and each button click is called a *query*. The answer to the query is presented graphically. Experimental results have shown that dynamic queries are a fast, effective, fun, and easy-to-use tool for novice and expert users to find trends and spot exceptions [2, 16].

Dynamic query user interfaces apply the principles of direct manipulation to query formulation and provide:

– a visual representation of the query and the results,
– rapid, incremental, and reversible actions,
– selection by pointing (not typing), and
– immediate and continuous display of results.

Some demos of dynamic queries are available from [10, 12].

1.3 What are Query Previews?

In a networked information system, there are three major obstacles facing users in a querying process: slow network performance, large data volume, and data complexity. EOSDIS has all of these. The collection is predicted to reach into the petabytes (10^{15} bytes). Additionally, EOSDIS datasets have numerous attributes, such as when and where it was collected, and the types of features or measurements in the dataset. With a forms-based interface, finding a particular dataset or a group of datasets that match certain characteristics would typically involve several iterations of querying and waiting for results over the network. This would not only slow down the process of finding datasets, but would slow down the network and servers for all users.

Query previews, as developed in [4], take advantage of the fact that in many cases, perhaps most, users are only interested in a small subset of the entire collection. For example, a user might only be interested in data for Europe, instead of the whole world. Narrowing the scope of the collection can greatly improve the efficiency of browsing and querying. A query previewer gives the user overviews of the entire collection such as a map showing the distribution of datasets over the Earth. The total number of datasets is also displayed. Using a dynamic-query interface, the user can narrow his search to a selected region of latitude and longitude by adjusting sliders. With another slider, the user can narrow his search to a certain range of years. With checkboxes, the user can narrow his search to only those datasets containing selected attributes (for example, temperature and pressure). As search parameters are adjusted (query), the distribution of datasets and the total count are updated.

Once the scope of the search is sufficiently narrow, *i.e.*, the number of datasets matching the dynamic query is manageably small, the user and the system are ready for more detailed query and exploration. This second query phase, called "query refinement," is another dynamic-query interface to a more detailed view of the datasets selected by the query preview. The details of query refinement are independent of the query previewer and will not be addressed in this paper.

Practical Requirements. Queries will be answered via a computation that consults a table of summary information about all datasets. Since the query previewer is a dynamic-query interface, updates should ideally appear to be continuous. Studies suggest that most users will in fact tolerate a delay of at most 0.1 or 0.2 seconds per update [1]. Since even this is much slower than typical world-wide-web turnaround, it is necessary that the aforementioned table be stored on the user's own node. Thus, disk-space limitations and download times both dictate that the table not be very large; 1 megabyte seems like a good rule of thumb. To summarize, we need small tables that support fast querying.

2 The Query-Previewing Problem

In the EOSDIS example, each dataset is described by a 4-dimensional record whose fields indicate the scope of information that the dataset contains: (1) range of latitude, (2) range of longitude, (3) range of years, and (4) set of attributes. The user's query is also a record of this type. An EOSDIS dataset matches the record if its range of latitudes overlaps with the user's range of latitudes, and so on for the other three dimensions. The result of the query is the number of EOSDIS datasets that match the query.

In order to compute the distribution of datasets over the Earth, the map is partitioned into squares and one query of the type described above is evaluated for each square in order to determine the number of datasets with information about that region of the Earth. Because we will, in fact, give a constant-time querying algorithm, the partition of the Earth into squares can in practice be very fine.

2.1 Geometric Interpretation

The first three fields in a record are ranges of numbers, so they can be represented as intervals. Let us ignore temporarily the fourth field (set of attributes). Then the record can be represented as a 3-dimensional rectangle, the Cartesian (cross) product of those three intervals. An EOSDIS record matches the query if the two corresponding rectangles overlap, i.e, if their intersection is nonempty. Please forgive us for belaboring an obvious point: rectangle overlap is a logical AND, *i.e.*, two rectangles overlap if the intervals along the first dimension overlap and the intervals along the second dimension overlap and the intervals along the third dimension overlap.

Let us return to the fourth field in our EOSDIS records, the set of attributes. We will represent each attribute by a number, so the fourth field is a set of numbers. At the risk of forcing the geometric metaphor, we will call a set of numbers a *generalized interval.*

Because the Earth is round, it may also be necessary to consider intervals in dimension (2) that wrap around from the right edge of the map to the left edge. These are called wrapped intervals.

2.2 Formal Problem Statements

In general we will consider records with d fields, giving rise to d-dimensional problems.

Definition 1.
- $\mathsf{N} = \{0, 1, \ldots\}$, the set of natural numbers
- N^d is the set of all d-dimensional lattice points in the 1st quadrant
- An *interval* is a set $\{a, a+1, \ldots, b\}$ of consecutive natural numbers.
- A *wrapped interval* in $\{1, \ldots, m\}$ is either an interval or the union of two intervals that contain 1 and m.
- A *generalized interval* is a subset of N.
- A *rectangle* in N^d is a cross-product $I_1 \times \cdots \times I_d$ of d intervals $I_1, \ldots, I_d$.
- A *wrapped rectangle* in N^d is a cross-product $I_1 \times \cdots \times I_d$ of d wrapped intervals $I_1, \ldots, I_d$.
- A *generalized rectangle* in N^d is a cross-product $I_1 \times \cdots \times I_d$ of d generalized intervals $I_1, \ldots, I_d$.
- *A intersects B* if $A \cap B \neq \emptyset$.
- *A is skew to B* if there is no hyperplane parallel to the coordinate axes that intersects both A and B.

In this paper, we will be mainly interested in three problems:

Rectangle Intersection (logical AND).

Data to be preprocessed: A list D of rectangles in N^d
Problem Instance: A single rectangle Q in N^d
Question: How many elements of D intersect Q?

Wrapped Rectangle Intersection (logical AND).

Data to be preprocessed: A list D of wrapped rectangles in $\{1, \ldots, m\}^d$
Problem Instance: A single wrapped rectangle Q in $\{1, \ldots, m\}^d$
Question: How many elements of D intersect Q?

Generalized Rectangle Intersection (logical AND).

Data to be preprocessed: A list D of generalized rectangles in N^d
Problem Instance: A single generalized rectangle Q in N^d
Question: How many elements of D intersect Q?

Although we do not state it as a formal problem, in practice we are interested in the mixed case, where some dimensions of the records are intervals, others are wrapped intervals, and yet others are generalized intervals. Our results for the homogeneous problems described above are directly applicable to the mixed case.

The following problem corresponds to queries based on logical OR rather than logical AND. Because such queries are sometimes useful in browsing, we consider them as well.

Rectangle Nonskewness (logical OR).

> **Data to be preprocessed:** A list D of rectangles in N^d
> **Problem Instance:** A single rectangle Q in N^d
> **Question:** How many elements of D are not skew to Q?

2.3 Complexity Bounds

Throughout, let R denote a fixed rectangle in N^d that contains each element of D. We present algorithms for rectangle intersection and rectangle nonskewness that use tables whose size depends only on the dimension d and the bounding rectangle R, and answer queries in time that depends only on d. The preprocessing time depends on D, but the cost for adding a single dataset to the list depends only on d and R.

Results. Assume that R is an $n_1 \times \cdots \times n_d$ rectangle. Let

$$\rho \;=\; (2n_1 - 1) \cdots (2n_d - 1) < 2^d |R|.$$

- Rectangle Intersection can be solved with $O(\rho)$ preprocessing per element of D, using tables of size ρ, in time $O(d2^d)$ per query.
- Rectangle Nonskewness can be solved with $O(\rho)$ preprocessing per element of D, using tables of size ρ, in time $O(d4^d)$ per query.
- Generalized Rectangle Intersection requires exhaustive tables (size $2^{n_1}2^{n_2}\cdots 2^{n_d}$).
- Wrapped Interval Intersection requires exhaustive tables (size $n_1(n_1-1)n_2(n_2-1)\cdots n_d(n_d-1)$).

3 Geometric Algorithms

We identify each rectangle in N^d with the polytope obtained upon replacing each of its points p with a unit d-cube centered at p. A face of a bounded-polytope S is *interior* to S if it is not the "exterior face" and it is not entirely contained in the boundary of S. Let $F_k(S)$ denote the number of k-dimensional faces of S and let $F_k^*(S)$ denote the number of k-dimensional faces interior to S.

Lemma 2. *If S is a bounded, connected d-dimensional polytope then*

$$\sum_{0 \le k \le d} (-1)^{d-k} F_k^*(S) = 1.$$

Proof. By Euler's theorem (see, for example, [9]),

$$\sum_{0 \le k \le d} (-1)^k F_k(S) = 1 + (-1)^d.$$

Let B denote the boundary of S. Then B is a connected $(d-1)$-dimensional polytope, so by Euler's theorem

$$\sum_{0\le k\le d-1} (-1)^k F_k(B) = 1 + (-1)^{d-1}.$$

Therefore

$$\begin{aligned}\sum_{0\le k\le d} (-1)^k F_k^*(S) &= \sum_{0\le k\le d} (-1)^k F_k(S) - (-1)^d - \sum_{0\le k\le d-1} (-1)^k F_k(B)\\ &= 1 + (-1)^d - (-1)^d - (1 + (-1)^{d-1})\\ &= -(-1)^{d-1}\\ &= (-1)^d\end{aligned}$$

so $\sum_{0\le k\le d} (-1)^{d-k} F_k^*(S) = (-1)^d(-1)^d = 1$. □

For each 0-, 1-, ..., or d-dimensional cube c, let $\#(c)$ denote the number of rectangles r in the list D such that the interior of r intersects the interior of c. In particular, the answer to the query Q is $\#(Q)$. We have

$$\#(Q) = \sum_{0\le k\le d} (-1)^{d-k} \sum_{c \text{ is a } k\text{-dimensional unit cube in the interior of } Q} \#(c) \tag{1}$$

Why? Because each rectangle in D that does not intersect Q contributes 0 to the sum, and each rectangle in D that intersects Q contributes 1 to the sum.

Let $\dim(r)$ denote the dimension of a rectangle. If we stored $(-1)^{d-\dim(c)}\#(c)$ for each unit cube c, then we could compute the sum specified in Equation (1) by summing over each unit cube contained in Q. Better yet, as noted in [3], if we store d-dimensional prefix sums, we can evaluate that sum in constant time. For each unit cube a we store $\sum_{b\le a} (-1)^{d-\dim(b)}\#(b)$, where the inequality must hold on every coordinate. Given such a table, the sum specified in Equation (1) may be obtained with $2^d - 2$ additions and subtractions, by the principle of inclusion and exclusion. The total storage needed is the number of unit cubes interior to R whose dimension is d or less, which is exactly ρ.

For concreteness we present the table construction and the query algorithm for the case $d = 2$ (Figure 1). In order to simplify the algorithm, we have used a table of size $2^d|R|$, which is larger than we claimed. This allows us to store 0s along the edges and avoid special cases.

3.1 Rectangle Nonskewness

By the principle of inclusion and exclusion, rectangle nonskewness is reduced to $2^d - 1$ instances of rectangle intersection. Therefore it can be solved with exactly the same table, in time $O(d4^d)$.

```
procedure CountRecord($x_1, x_2, y_1, y_2$)
    for $i = 2x_1 - 1$ to $2x_2 - 1$ do
        for $j = 2y_1 - 1$ to $2y_2 - 1$ do
            table$[i, j]$ = table$[i, j] + (-1)^{i+1}(-1)^{j+1}$
end

procedure CountAllRecords
    for $i = 0$ to $2n_1 - 1$ do
        for $j = 0$ to $2n_2 - 1$ do
            table$[i, j] = 0$
    for each rectangle $(x_1, x_2, y_1, y_2)$ in the list $D$ do
        CountRecord($x_1, x_2, y_1, y_2$)
end

procedure ComputePartialSums
    for $i = 2$ to $2n_1 - 1$ do
        for $j = 1$ to $2n_2 - 1$ do
            table$[i, j]$ = table$[i - 1, j]$ + table$[i, j]$
    for $i = 1$ to $2n_1 - 1$ do
        for $j = 2$ to $2n_2 - 1$ do
            table$[i, j]$ = table$[i, j - 1]$ + table$[i, j]$
end

procedure BuildTable
    CountAllRecords
    ComputePartialSums
end

function IncludeExclude($a_1, a_2, b_1, b_2$)
    return table$(a_2, b_2)$ − table$(a_2, b_1)$ − table$(a_1, b_2)$ + table$(a_1, b_1)$
end

function Query($x_1, x_2, y_1, y_2$)
    return IncludeExclude($2x_1 - 2, 2x_2 - 1, 2y_1 - 2, 2y_2 - 1$)
end
```

Fig. 1. Table Construction and Query Algorithm for the case $d = 2$.

4 Lower Bounds

There are exactly 2^n generalized intervals and exactly $n(n-1)$ wrapped intervals in $\{1, \ldots, n\}$. Therefore, Generalized Rectangle Intersection can be solved by table lookup with a table of size $2^{n_1} \cdots 2^{n_d}$, and Wrapped Interval Section can be solved by table lookup with a table of size $n_1(n_1 - 1) \cdots n_d(n_d - 1)$. We will show that no smaller tables suffice for either problem, no matter how much time is allowed.

For simplicity we will consider only the case $d = 1$. In the full version of this paper we will explain how the general case is a corollary of this one. Henceforth let $n = n_1$.

4.1 Generalized Rectangle Intersection

Suppose that given some table T we can answer questions of the form "how many elements of D intersect Q?", where D is a fixed multiset of generalized intervals and Q is a generalized interval. Previously, we said that the intersection questions are really logical-AND questions. Actually, they are ANDs over all dimensions. But in each single dimension, the question is an OR, i.e., "does at least one square of the query interval belong to the dataset rectangle?"

Let $\vee$ denote logical OR, and $\wedge$ denote logical AND. Let $\#(x_1 \vee \cdots \vee x_k)$ denote the number of generalized rectangles r in D such that $x_1 \in r \vee \cdots \vee x_k \in r$. Let $\#(x_1 \wedge \cdots \wedge x_k)$ denote the number of generalized rectangles r in D such that $x_1 \in r \wedge \cdots \wedge x_k \in r$. By assumption, we can compute $\#(x_1 \vee \cdots \vee x_k)$ from T. By the principal of inclusion and exclusion, we have

$$\#(x_1 \wedge x_2) = \#(x_1) + \#(x_2) - \#(x_1 \vee x_2).$$

Thus we can compute $\#(x_1 \wedge x_2)$ from T. By a simple induction, we can compute $\#(x_1 \wedge \cdots \wedge x_k)$ from T.

Let $\#(x_1 \wedge \cdots \wedge x_k \wedge \neg x_{k+1} \wedge \cdots \wedge \neg x_m)$ denote the number of generalized rectangles r in D such that $x_1 \in r \wedge \cdots \wedge x_k \in r \wedge x_{k+1} \notin r \wedge \cdots \wedge x_m \notin r$. We have

$$\#(x_1 \wedge \cdots \wedge x_k \wedge \neg x_{k+1}) = \#(x_1 \wedge \cdots \wedge x_k) - \#(x_1 \wedge \cdots \wedge x_{k+1}).$$

Thus we can compute $\#(x_1 \wedge \cdots \wedge x_k \wedge \neg x_{k+1})$ from T. By a simple induction we can compute $\#(x_1 \wedge \cdots \wedge x_k \wedge \neg x_{k+1} \wedge \cdots \wedge \neg x_n)$ from T, where $\{x_1, \ldots, x_n\} = \{1, \ldots, n\}$. Thus we can determine from T exactly how many times the generalized rectangle $\{x_1, \ldots, x_k\}$ appears in the multiset D. Since we can recover 2^n independent numbers from T, the size of T must be at least 2^n.

Note: a similar argument shows that if we limit the size of generalized intervals to k then we still can't get by with nonexhaustive tables.

4.2 Wrapped Rectangle Intersection

Suppose that given some table T we can answer questions of the form "how many elements of D intersect Q?", where D is a fixed multiset of wrapped intervals and Q is a wrapped interval. If $i \le j$, let $\#[i, j]$ denote the number of elements of D contained in the interval $[i, j]$. Then $\#[i, j]$ is equal to the number of elements of D (that intersect $[1, n]$) minus the number of elements of D that intersect $\{j+1, \ldots, n, 1, \ldots, i-1\}$, so we can compute $\#[i, j]$ from T.

The number of times that the interval $\#[i, j]$ appears in the list D is given by the formula:

$$\#[i, j] - \#[i-1, j] - \#[i, j-1] + \#[i-1, j-1].$$

By a similar argument we can determine the number of times each wrapped interval appears in D. Since we can recover $n(n-1)$ independent numbers from T, the size of T must be at least $n(n-1)$.

Acknowledgments

This work was supported by NASA grant 52895. The research was performed while the first author was on sabbatical from Yale University, visiting the Human–Computer Interaction Laboratory at the University of Maryland. He is also partially supported by NSF under grants CCR-9700417 and CCR-9796317. Both authors are grateful to Catherine Plaisant and Ben Shneiderman for their part in formulating this problem and to Dave Mount and Dan Spielman for helpful discussions.

References

1. Ahlberg, C. and Shneiderman, B., Visual Information Seeking: Tight Coupling of Dynamic Query Filters with Starfield Displays, *Proc. ACM SIGCHI* (1994) 313–317
2. Ahlberg, C. and Wistrand, E., IVEE: An Information Visualization and Exploration Environment, *Proc. IEEE Info. Vis.*, (1995) 66–73
3. Bestul, T., Parallel paradigms and practices for spatial data, Ph.D. Thesis, Univ. Maryland Dept. Comp. Sci., TR-2897, (1992)
4. Doan, K., Plaisant, C., and Shneiderman, B., Query Previews in Networked Information Systems, *Proc. Forum Adv. Digit. Libr.*, *IEEE Comp. Soc. Press*, (1996) 120–129
5. Eick, S., Data Visualization Sliders, *Proc. User Interf. Softw. Techn.* (1994) 119–120
6. HCIL, http://www.cs.umd.edu/projects/hcil/Research/1995/dq-for-eosdis.html
7. Fishkin, K. and Stone, M. C., Enhanced Dynamic Queries via Movable Filters, *Proc. ACM SIGCHI* (1995) 415–420
8. Goldstein, J. and Roth, S. F., Using Aggregation and Dynamic Queries for Exploring Large Data Sets, *Proc. ACM SIGCHI* (1994) 23–29
9. Harary, F., *Graph Theory*, Addison–Wesley (1969)
10. HCIL, ftp://ftp.cs.umd.edu/pub/hcil/Demos/DQ/dq-home.zip. Down-loadable PC demo
11. Ioannidis, Y., Dynamic Information Visualization, *ACM SIGMOD Rec.*, **25** (1996) 16–20

12. Information Visualization and Exploration Environment (IVEE) Development AB, http://www.ivee.com/. Online Java demo and down-loadable demos for various platforms
13. Shneiderman, B., Dynamic Queries for Visual Information Seeking, *IEEE Softw.*, **11** (1994) 70–77
14. Shneiderman, B., Racing to the winning line with visual data mining, http://www.ivee.com/corporate/columns/race.html
15. Tanin, E., Beigel, R., and Shneiderman, B., Incremental Data Structures and Algorithms for Dynamic Query Interfaces, *ACM SIGMOD Rec.*, **25** (1996) 21–24
16. Williamson, C. and Shneiderman, B., The Dynamic HomeFinder: Evaluating Dynamic Queries in a Real-Estate Information Exploration System, *Proc. ACM SIGIR* (1992) 339–346