# Efficient Spatial Sampling of Large Geographical Tables

Anish Das Sarma, Hongrae Lee, Hector Gonzalez, Jayant Madhavan, Alon Halevy
Google Research
Mountain View, CA, USA
{anish,hrlee,hagonzal,jayant,halevy}@google.com

## ABSTRACT

Large-scale map visualization systems play an increasingly important role in presenting geographic datasets to end users. Since these datasets can be extremely large, a map rendering system often needs to select a small fraction of the data to visualize them in a limited space. This paper addresses the fundamental challenge of *thinning*: determining appropriate samples of data to be shown on specific geographical regions and zoom levels. Other than the sheer scale of the data, the thinning problem is challenging because of a number of other reasons: (1) data can consist of complex geographical shapes, (2) rendering of data needs to satisfy certain constraints, such as data being preserved across zoom levels and adjacent regions, and (3) after satisfying the constraints, an *optimal* solution needs to be chosen based on *objectives* such as *maximality*, *fairness*, and *importance* of data.

This paper formally defines and presents a complete solution to the thinning problem. First, we express the problem as an integer programming formulation that efficiently solves thinning for desired objectives. Second, we present more efficient solutions for maximality, based on DFS traversal of a spatial tree. Third, we consider the common special case of point datasets, and present an even more efficient randomized algorithm. Finally, we have implemented all techniques from this paper in Google Maps [6] visualizations of Fusion Tables [14], and we describe a set of experiments that demonstrate the tradeoffs among the algorithms.

## Categories and Subject Descriptors

H.0 [**Information Systems**]: General—*storage, retrieval*; H.2.4 [**Database Management**]: Systems

## General Terms

Algorithms, Design, Management, Performance

## Keywords

geographical databases, spatial sampling, maps, data visualization, indexing, query processing


*SIGMOD '12,* May 20–24, 2012, Scottsdale, Arizona, USA.


## 1. INTRODUCTION

Several recent cloud-based systems try to broaden the audience of database users and data consumers by emphasizing ease of use, data sharing, and creation of map and other visualizations [2, 3, 5, 8, 14]. These applications have been particularly useful for journalists embedding data in their articles, for crisis response where timely data is critical for people in need, and are becoming useful for enterprises with collections of data grounded in locations on maps [11].

Map visualizations typically show data by rendering *tiles* or *cells* (rectangular regions on a map). One of the key challenges in serving data in these systems is that the datasets can be huge, but only a small number of records per cell can be sent to the browser at any given time. For example, the dataset including all the house parcels in the United States has more than 60 million rows, but the client browser can typically handle only far fewer (around 500) rows per cell at once. This paper considers the problem of *thinning* geographical datasets: given a geographical region at a particular zoom level, return a small number of records to be shown on the map.

In addition to the sheer size of the data and the stringent latency requirements on serving the data, the thinning problem is challenging for the following reasons:

- In addition to representing points on the map, the data can also consist of complex polygons (e.g., a national park), and hence span multiple adjacent map cells.
- The experience of zooming and panning across the map needs to be seamless, which raises two constraints:
  - Zoom Consistency: If a record $r$ appears on a map, further zooming into the region containing $r$ should not cause $r$ to disappear. In other words, if a record appears at any coarse zoom granularity, it must continue to appear in all finer granularities of that region.
  - Adjacency: If a polygon spans multiple cells, it must either appear in all cells it spans or none; i.e., we must maintain the geographical shape of every record.

Figure 1 demonstrates an example of zoom consistency violation. In Figure 1(a), suppose the user wants to zoom in to see more details on the location with a ballon icon. It would not be natural if further zoom-in makes the location disappear as in Figure 1(b). Figure 2 shows an example of adjacency consistency violation for polygons. The map looks broken because the display of polygons that span multiple cells is not consistent.

Even with the above constraints, there may still be multiple different sets of records that can be shown in any part


(a) Original viewpoint. (b) Violation of zoom consistency (c) Correct zoom in

**Figure 1: Violation of Zoom Consistency**

**Figure 2: Violation of Adjacency Constraint**

of the region. The determination of which set of points to show is made by application-specific *objective functions*. The most natural objective is "maximality", i.e., showing as many records as possible while respecting the constraints above. Alternatively, we may choose to show records based on some notion of "importance" (e.g., rating of businesses), or based on maximizing "fairness", treating all records equally.

This paper makes the following contributions. First, we present an integer programming formulation of size linear in the input that encodes constraints of the thinning problem and enables us to capture a wide variety of objective functions. We show how to construct this program, capturing various objective criteria, solve it, and translate the program's solution to a solution of the thinning problem.

Second, we study in more detail the specific objective of *maximality*: we present notions of *strong* and *weak maximality*, and show that obtaining an optimal solution based on strong maximality is NP-hard. We present an efficient DFS traversal-based algorithm that guarantees weak maximality for any dataset, and strong maximality for datasets with only point records.

Third, we consider the commonly occurring special case of datasets that only consist of points. We present a randomized algorithm that ensures strong maximality for points, and is much more efficient than the DFS algorithm.

Finally, we describe a detailed experimental evaluation of our techniques over large-scale real datasets in Google Fusion Tables [14]. The experiments show that the proposed solutions efficiently select records respecting aforementioned constraints.

Section 7 discusses the related area of cartographic generalization, and presents other related work. The rest of the paper is organized as follows. Section 2 defines the thinning problem formally. Section 3 describes the integer programming solution to the thinning problem. Section 4 studies in detail maximality for arbitrary regions, and Section 5 looks at the special case of datasets with point regions. Experiments are presented in Section 6, and we conclude in Section 8. Due to space constraints, proofs for technical results are omitted.

## 2. DEFINITIONS

We begin by formally defining our problem setting, starting with the spatial organization of the world, defining regions and geographical datasets (Section 2.1), and then formally defining the thinning problem (Section 2.2).

### 2.1 Geographical data

#### *Spatial Organization*

To model geographical data, the world is spatially divided into multiple *cells*, where each cell corresponds to a region of the world. Any region of the world may be seen at a specific *zoom level* $z \in [1, \mathcal{Z}]$, where 1 corresponds to the coarsest zoom level and $\mathcal{Z}$ is the finest granularity. At zoom level 1, the entire world fits in a single cell $c_1^1$. At zoom level 2, $c_1^1$ is divided into four disjoint regions represented by cells $\{c_1^2, \ldots, c_4^2\}$; zoom 3 consists of each cell $c_i^2$ further divided into four cells, giving a set of 16 disjoint cells $c_1^3, \ldots, c_{16}^3$, and so on. Figure 1(a) is a cell at $z = 13$, and Figures 1(b) and (c) are cells at $z = 14$. In general, the entire spatial region is hierarchically divided into multiple regions as defined by the tree structure below.

DEFINITION 2.1 (SPATIAL TREE). *A spatial tree $T(\mathcal{Z}, \mathcal{N})$ with a maximum zoom level $\mathcal{Z} \geq 1$ is a balanced 4-ary rooted tree with $\mathcal{Z}$ levels and nodes $\mathcal{N}$, with $4^{\mathcal{Z}-1}$ nodes at level-$\mathcal{Z}$ denoted $N^{\mathcal{Z}} = \{c_1^{\mathcal{Z}}, \ldots, c_{4^{\mathcal{Z}}-1}^{\mathcal{Z}}\}$.*

The nodes at each level of the tree correspond to a complete and disjoint cell decomposition of an entire region, represented as one cell at the root. Values of $\mathcal{Z}$ in most commercial mapping systems range between 10 and 20 (it is 20 for Google Maps [6]).

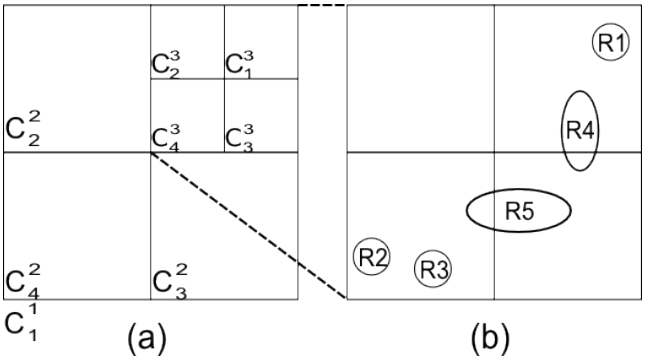



**Figure 3: Running Example: (a) Spatial tree with $\mathcal{Z} = 3$; (b) Regions shown at $z = 3$ for $c_1^2$.**

EXAMPLE 2.1. *Figure 3(a) shows a spatial organization of a tree with $\mathcal{Z} = 3$. At zoom-level $z = 1$ the entire space is a single cell, which are divided into 4 cells at $z = 2$, and 16 at the finest zoom level of $z = 3$. (The figure only shows the $z = 3$ cells for the cell $c_1^2$ at $z = 2$.)*

Note that such a hierarchical division of a region into subregions corresponds to a space-filling curve [26]. Thus, the nodes at a particular level in the spatial tree can be used for index range scans for a subregion, when ordered based on the space-filling curve.

### *Regions and Span*

A region corresponds to a part of the world. Since the finest granularity of data corresponds to cells at zoom level $\mathcal{Z}$, any region can be defined by a subset of cells at zoom level $\mathcal{Z}$.

DEFINITION 2.2 (REGION AND POINT REGION). *A region $R(S)$ over a spatial tree $T(\mathcal{Z}, \mathcal{N})$ is defined by a subset $S \subseteq \mathcal{N}^{\mathcal{Z}}$, $|S| \geq 1$. A region $R(S)$ is said to be a* point region *iff $|S| = 1$.*

We often refer to regions that span cells at different levels:

DEFINITION 2.3 (REGION SPAN). *A region $R(S)$ over spatial tree $T(\mathcal{Z}, \mathcal{N})$ is said to* span *a cell $c_i^z \in \mathcal{N}$ iff $\exists c_j^{\mathcal{Z}} \in \mathcal{N}^{\mathcal{Z}}$ such that $c_j^{\mathcal{Z}} \in S$ and $c_i^z$ is an ancestor of $c_j^{\mathcal{Z}}$ in $T$. We use span$(R)$ to denote the set of all cells $R$ spans.*

Note that a region defined by a set of finest-granularity cells in the maximum zoom level spans every ancestor cell of these finest-granularity cells.

EXAMPLE 2.2. *Figure 3(b) shows 5 regions for the cell $c_1^2$, showing their spans at $z = 3$ over cells $c_1^3, \ldots, c_4^3$. Regions $R1$, $R2$, and $R3$ are point regions spanning only a single cell at $z = 3$ (and three cells each across the three zoom levels), and $R4$ and $R5$ span two cells at $z = 3$ (and 4 cells in aggregate: two each at $z = 3$ and one each at $z = 1, 2$).*

### *Geographical Dataset*

A geographical dataset (*geoset*, for short) consists of a set of records, each describing either a point or a polygon on a map. For the purposes of our discussion it suffices to consider the regions occupied by the records. Specifically, (1) a record describing a point can be represented by a point region, and (2) a record describing a polygon $p$ can be represented by the region defined by set of finest-granularity regions in $\mathcal{N}^{\mathcal{Z}}$ that $p$ occupies. In practice, we represent the actual points and polygons in addition to other structured data associated with the location (e.g., restaurant name, phone number).

DEFINITION 2.4 (GEOSET). *A geoset $G = \{R_1, \ldots, R_n\}$ over spatial tree $T(\mathcal{Z}, \mathcal{N})$ is a set of $n$ regions over $T$ corresponding to $n$ distinct records. $R_i$ represents the region of the record with identifier $i$.*

## 2.2 The thinning problem

We are now ready to formally introduce the thinning problem. We start by describing the constraints that a solution to thinning must satisfy (Section 2.2.1), and then motivate some of the objectives that go into picking one among multiple thinning solutions that satisfy the constraints (Section 2.2.2).

### *2.2.1 Constraints*

To provide a seamless zooming and panning experience on the map, a solution to the thinning problem needs to satisfy the following constraints:

1. **Visibility:** The number of visible regions at any cell $c_i^z$ is bounded by a fixed constant $K$.
2. **Zoom Consistency:** If a region $R$ is visible at a cell $c_i^z$, it must also be visible at each descendant cell $c_{i'}^{z'}$ of $c_i^z$ that is spanned by $R$. The reason for this constraint is that as a user zooms into the map she should not lose points that are already visible.
3. **Adjacency:** If a region $R$ is visible at a cell $c_i^z$, it must also be visible at each cell $c_{i'}^z$ spanned by $R$. This constraint ensures that each region is visible in its entirety when moving a map around (at the same zoom level), and is not "cut out" from some cells and only partially visible. Note that adjacency is trivial for points but not for polygons.

EXAMPLE 2.3. *Going back to the data from Figure 3, suppose we have a visibility bound of $K = 1$, then at most one of $R1 - R5$ can be visible in $c_1^1$, one of $R1, R4$ can be visible at $c_1^3$, and at most one of $R2 - R5$ can be visible in cell $c_3^3$. Based on the zoom consistency constraint, if $R4$ is visible in $c_1^1$, then it must be visible in $c_1^2$, $c_1^3$, and $c_3^3$. The adjacency constraint imposes that $R5$ is visible in neither or both of $c_3^3$ and $c_4^3$.*

A consequence of the zoom consistency and adjacency constraints is that every region must be visible at all spanned cells starting at some particular zoom level. We can therefore define thinning as the problem of finding the initial zoom level at which each record becomes visible.

PROBLEM 2.1 (THINNING). *Given a geoset $G = \{R_1, \ldots, R_n\}$ over a spatial tree $T(\mathcal{Z}, \mathcal{N})$, and a maximum bound $K \in \mathbb{N}$ on the number of visible records in any cell, compute a function* min-level $M : \{1, \ldots, n\} \rightarrow \{1, \ldots, \mathcal{Z}, \mathcal{Z} + 1\}$ *such that the following holds:*

> **Visibility Bound:** *$\forall c_j^z \in \mathcal{N}$, $z \leq \mathcal{Z}$, we must have $|Vis_M(G, T, c_j^z)| \leq K$, where $Vis_M(G, T, c_j^z)$ denotes the set of all visible records at cell $c_j^z$ whose min-level is set to at most $z$:*
>
> $$Vis_M(G, T, c_j^z) = \{R_i | (c_j^z \in span(R_i)) \& (M(j) \leq z)\}$$

Intuitively, the min-level function assigns for each record the coarsest-granularity zoom level at which the record will start being visible and continue to be visible in all finer granularities. (A min-level of $\mathcal{Z}+1$ means that record is never visible.) By definition, assigning a single min-level for each record satisfies the *Zoom Consistency* property. Further, the fact that we are assigning a single zoom level for each record imposes the condition that if a record is visible at one spanned cell at a particular level, it will also be visible at all other spanned cells at the same level. Thus, the *Adjacency* property is also satisfied. The first condition in the problem above ensures that at any specific cell in the spatial tree $T$, at most a pre-specified number $K$ of records are visible.

EXAMPLE 2.4. *Considering the data from Figure 3, with $K = 1$, we have several possible solutions to the thinning solution. A trivial function $M^1(Ri) = 4$ is a solution that doesn't show any region on any of the cells. A more interesting solution is $M^2(R1) = 1$, $M^2(R2) = 3$, and $M^2(\cdot) = 4$ for all other regions. This solution shows $R1$ in its cell from $z = 1$ itself, and $R2$ from $z = 3$. Another solution $M^3$ is obtained by setting $M^3(R1) = 2$ above and $M^3(\cdot)$ being identical to $M^2(\cdot)$ for other regions; $M^3$ shows $R1$ only starting at $z = 2$. Arguably, $M^2$ is "better" than $M^3$ since $R1$ is shown in more cells without compromising the visibility of any other region; next we discuss this point further.*

### 2.2.2 Objectives

There may be a large number of solutions to the thinning problem that satisfy the constraints described above, including the trivial and useless one setting the min-level of every region to $\mathcal{Z}+1$. Below we define informally certain desirable *objective functions*, which can be used to guide the selection of a specific solution. In the next section we describe a thinning algorithm that enables applying these objectives.

1. **Maximality:** Show as many records as possible in any particular cell, assuming the zoom consistency and adjacency properties are satisfied.
2. **Fairness:** Ensure that every record has some chance of being visible in a particular cell, if showing that record doesn't make it impossible to satisfy the constraints.
3. **Region Importance:** Select records such that more "important" records have a higher likelihood of being visible than less important ones. For instance, importance of restaurants may be determined by their star rating, and if there are two restaurants in the same location, the one with the higher rating should have a greater chance of being sampled.

Not surprisingly, these objectives may conflict with one another, as shown by our next example. We can define several other intuitive objectives not considered above (e.g., respecting "spatial density"); a comprehensive study of more objectives is left as future work.

EXAMPLE 2.5. *Continuing with our data from Figure 3 and thinning solutions from Example 2.4, clearly $M^1$ is not maximal. We shall formally define maximality later, but it is also evident that $M^3$ is not maximal, as $M^2$ shows a strictly larger number of records. Fairness would intuitively mean that if possible every record should have a chance of being visible; furthermore, regions that have identical spans (e.g., $R2$ and $R3$) should have equal chance of being visible. Finally, if we consider some notion of importance, and suppose $R2$ is much more important than $R3$, then $R2$ should have a correspondingly higher likelihood of being visible.*

## 2.3 Outline of our solutions

In Section 3 we show how to formulate the thinning problem as an integer programming problem in a way that expresses the different objectives we described above. In Section 4, we consider the maximality objective in more detail and show that while one notion of maximality renders the thinning problem NP-hard, there is a weaker form of maximality that enables an efficient solution. Finally, in Section 5, we study the special case of a geoset consisting of point records only.

We note that this paper considers a query-independent notion of thinning, which we can compute off-line. We leave query-dependent thinning to future work, but note that zooming and panning an entire dataset is a very common scenario in practice. We also note that a system for browsing large geographical datasets also needs to address challenges that are not considered here such as simplification of arbitrary polygons in coarser zoom levels and dynamic styling of regions based on attribute values (e.g., deciding the color or shape of an icon).

# 3. THINNING AS INTEGER PROGRAMMING

In this section we describe an integer program that combines various objectives from Section 2.2 into the thinning problem. Section 3.1 describes the construction of the integer program and Section 3.2 discusses solving it.

## 3.1 Constructing the integer program

### 3.1.1 Modeling constraints

Given an instance of the thinning problem, i.e., a geoset $G = \{R_1, \ldots, R_n\}$ over a spatial tree $T(\mathcal{Z}, \mathcal{N})$, and a maximum bound $K \in \mathbb{N}$ on the number of visible records in any cell, we construct an integer program $\mathbb{P}$ as follows (we refer to the construction algorithm by CPALGO):

**Partition the records based on spans:** We partition $G$ into equivalence classes $\mathcal{P}(G) = \{P_1, \ldots, P_l\}$ such that: (a) $\cup_{q=1}^{n} P_q = G$; and (b) $\forall q, \forall R_i, R_j \in P_q : span(R_i) = span(R_j)$. For ease of notation, we use $span(P_q)$ to denote the span of a record in $P_q$. These partitions are created easily in a single pass of the dataset by hashing the set of cells spanned by each record.

**Variables of the integer program:** the set of variables $\mathcal{V}$ in the program $\mathbb{P}$ are obtained from the partitions generated above: For each partition $P_q$, we construct $\mathcal{Z}$ variables $v_q^1, v_q^2, \ldots, v_q^{\mathcal{Z}}$. Intuitively, $v_q^z$ represents the *number* of records from partition $P_q$ whose min-level are set to $z$.

**Constraints:** The set $\mathcal{C}$ of constraints are:

1. **Sampling constraints:**

$$|P_q| \geq \sum_{z=1}^{\mathcal{Z}} v_q^z \tag{1}$$

$$\forall q \forall z : v_q^z \geq 0 \tag{2}$$

$$\forall q \forall z : v_q^z \in \mathbb{Z} \text{ i.e., } v_q^z \text{ is an integer} \tag{3}$$

Equation (1) ensures that the number of records picked for being visible at each zoom level does not exceed the total number of records in the partition. Further, $(|P_q| - \sum_{z=1}^{\mathcal{Z}} v_q^z)$ gives the number of records from $P_q$ that are not visible at any zoom level. Equations (2) and (3) simply ensure that only a positive integral number of records are picked from each partition from each zoom level. (Later we shall discuss the removal of the integer constraint in Equation (3) for efficiency.) Note that given a solution to the integer program we may sample any set of records from each partition $P_q$ respecting the solution.

2. **Zoom consistency and visibility constraint:** We have a visibility constraint for each cell that is spanned by at least one record:

$$\forall c_j^z \in \mathcal{N} : \sum_{q:c_j^z \in span(P_q)} \sum_{z^* \leq z} v_q^{z^*} \leq K \qquad (4)$$

The constraint above ensures that at cell $c_j^z$, at most $K$ records are visible. The expression on the left computes the number of records visible at $c_j^z$: for each partition $P_q$ spanning $c_j^z$, only and all variables $v_q^{z^*}$ correspond to visible regions. Note that all $v_q^{z^*}$ with $z^*$ strictly less than $z$ are also visible at $c_j^z$ due to the zoom consistency condition.

3. **Adjacency constraint:** we do not need to add another constraint because the adjacency constraint is satisfied by the construction of the variable $v_q^z$ itself: each region from $P_q$ visible at zoom level $z$ is visible at all cells spanned at level $z$.

**Producing the thinning solution:** Given a solution to the integer program, we produce a solution to the thinning problem by sampling without replacement for partition $P_q$ as follows. First we sample $v_q^1$ records from $P_q$ uniformly at random and set their $M$ value to 1, then sample $v_q^2$ records from the rest of $P_q$ and set their $M$ value to 2, and so on. The following theorem formally states the equivalence relationship of the constraints above to the thinning problem.

THEOREM 3.1. *Given a geoset $G = \{R_1, \ldots, R_n\}$ over a spatial tree $T(\mathcal{Z}, \mathcal{N})$, and a maximum bound $K \in \mathbb{N}$ on the number of visible records in any cell, the integer program $\mathbb{P}(\mathcal{P}, \mathcal{V}, \mathcal{C})$ constructed using Algorithm* CPALGO *above is an equivalent formulation of the thinning problem (Problem 2.1): $\mathbb{P}$ captures all and only solutions to the thinning problem. Furthermore, the size of the program satisfies $|\mathcal{V}| = \mathcal{Z}|\mathcal{P}| = \mathcal{O}(n\mathcal{Z})$ and $|\mathcal{C}| = \mathcal{O}(4^{\mathcal{Z}})$.*

### 3.1.2 Minimizing program size

The integer program created naively is exponential in the size of the input. We now present optimizations that reduce the number of variables and constraints using three key ideas: (1) Several partitions may be combined when the number of regions in a partition are small; (2) We only need to write the zoom consistency and visibility constraints (Equation (4) above) for *critical nodes*, which are typically far fewer than $4^{\mathcal{Z}}$; (3) Regions are typically described by a span of bounded size of say $M$ cells instead of any possible subset of the $\sim 4^{\mathcal{Z}}$ cells, therefore the total size of the input is bounded. All put together, we obtain an integer program that is linear in the size of the geoset (in terms of number of variables as well as the number of constraints).

**Algorithm 1** An algorithm for the construction of a merged partition $\mathcal{P}^m$ (inducing a smaller but equivalent integer programming solution) from the output of Algorithm CPALGO.

```
1: Input: (1) Geoset G = {R_1, ..., R_n} over spatial tree T(Z, N),
   visibility bound K ∈ N; (2) Output P, Cover(c), Touch(c) obtained from Algorithm CPALGO.
2: Output: Merged partitioning P^m.
3: Initialize P^m = P, Stack S = root(T) (i.e., the root node).
4: while S ≠ ∅ do
5:    Let node c = pop(S).
6:    // Check if c can be a valid merged partition root.
7:    if K ≥ Σ_{P∈Touch(c)} |P| then
8:       Construct merged partition P_c = ∪_{P∈Cover(c)} P.
9:       Set P^m = ({P_c} ∪ P^m) \ Cover(c).
10:   else
11:      if c is not leaf then
12:         Push each child of c into S.
```

### Merging Partitions

We show how the partitions $\mathcal{P}$ generated in Section 3.1.1 can be transformed to a merged partitioning $\mathcal{P}^m$ with fewer partitions while preserving all solutions of the original program. The integer program can be constructed with $\mathcal{P}^m$ as in Algorithm CPALGO. We denote the program induced by a partitioning $\mathcal{P}$ by $\mathbb{P}|_{\mathcal{P}}$. The following lemma specifies the required conditions from the merged partitioning.

LEMMA 3.1 (PARTITION MERGING). *Given a geoset $G = \{R_1, \ldots, R_n\}$ over a spatial tree $T(\mathcal{Z}, \mathcal{N})$, and a maximum bound $K \in \mathbb{N}$ on the number of visible records in any cell, the integer program $\mathbb{P}(\mathcal{P}, \mathcal{V}, \mathcal{C})$ over partitioning $\mathcal{P} = \{P_1, \ldots, P_l\}$, $\mathbb{P}|_{\mathcal{P}}$, is equivalent to the program $\mathbb{P}|_{\mathcal{P}^m}$ over a* merged partitioning *$\mathcal{P}^m = \{P_1^m, \ldots, P_{l^m}^m\}$ where the following hold:*

1. **Union:** *Each $P^m \in \mathcal{P}^m$ is a union of partitions in $\mathcal{P}$, i.e., $\forall P^m \in \mathcal{P}^m \exists S(P^m) \subseteq \mathcal{P} : P^m = \bigcup_{P \in S} P$*
2. **Disjoint Covering:** *For $P^m, P^n \in \mathcal{P}^m$, $m \neq n \Rightarrow (P^m \cap P^n = \emptyset)$; and $G = \bigcup_{P \in \mathcal{P}^m} P$*
3. **Size:** *Define $span(P^m) = \cup_{R_i \in P^m} span(S)$. Let the span of any partition of region restricted to nodes in zoom level $\mathcal{Z}$ be denoted $span_{\mathcal{Z}}$; i.e., $span_{\mathcal{Z}}(P) = span(P) \cap N^{\mathcal{Z}}$. Then the total number of records overlapping with $span_{\mathcal{Z}}$ of any merged partition is at most $K$: $\forall P^m \in \mathcal{P}^m : |\{R_i \in G | span_{\mathcal{Z}}(R_i) \cap span_{\mathcal{Z}}(P^m) \neq \emptyset\}| \leq K$.*

The intuition underlying Lemma 3.1 is that if multiple partitions in the original program cover at most $K$ records, then they can be merged into one partition without sacrificing important solutions to the integer program.

Algorithm 1 describes how to create the merged partitions. The algorithm uses two data structures that are easily constructed along with Algorithm CPALGO: (1) $Cover(c)$, $c \in \mathcal{N}$ returning all original partitions from $\mathcal{P}$ whose spanned leaf nodes are a subset of the leaf nodes descendant from $c$; (2) $Touch(c)$, $c \in \mathcal{N}$ returning all partitions from $\mathcal{P}$ that span some node in the subtree rooted at $c$. The algorithm constructs in a top-down fashion *subtree-partitions*, where each merged partition is responsible for all original partitions that completely fall under the subtree.

LEMMA 3.2. *Given geoset $G = \{R_1, \ldots, R_n\}$ over spatial tree $T(\mathcal{Z}, \mathcal{N})$, visibility bound $K \in \mathbb{N}$, and the output of Algorithm* CPALGO, *Algorithm 1 generates a merged partitioning $\mathcal{P}^m$ that satisfies the conditions in Lemma 3.1 and runs in one pass of the spatial tree.*

### Constraints Only on Critical Nodes

We now show how to reduce the number of constraints in the integer program by identifying *critical nodes* and writing constraints only for those nodes.

DEFINITION 3.1 (CRITICAL NODES). *Given a geoset $G = \{R_1, \ldots, R_n\}$ over a spatial tree $T(\mathcal{Z}, \mathcal{N})$, and a maximum bound $K \in \mathbb{N}$ on the number of visible records in any cell, and a set of (merged) partitions $\mathcal{P} = \{P_1, \ldots, P_l\}$ with corresponding spans of $span_{\mathcal{Z}}$ (as defined in Lemma 3.1), a node $c \in \mathcal{N}$ is said to be a* critical node *if and only if there exists a pair of nodes $c_{q_1} \in span_{\mathcal{Z}}(P_{q_1})$ and $c_{q_2} \in span_{\mathcal{Z}}(P_{q_2})$ such that $c$ is a least-common ancestor of $c_{q_1}, c_{q_2}$ in $T$.*

Intuitively, a node $c$ is a critical node if it is a least-common ancestor for at least two distinct partitions' corresponding cells. In other words, there are at least two partitions that meet at $c$, and no child of $c$ has exactly the same set of partition's nodes in their subtree. Clearly we can compute the set of critical nodes in a bottom up pass of the spatial tree starting with the set of (merged) partitions. Therefore, based on the assignment of values to variables in the integer program, the total number of regions visible at $c$ may differ from the number of nodes visible at parent/child nodes, requiring us to impose a visibility constraint on $c$. For any node $c'$ that is not a critical node, the total number of visible regions at $c'$ is identical to the first descendant critical node of $c'$, and therefore we don't need to separately write a visibility constraint at $c'$. Therefore, we have the following result.

LEMMA 3.3 (CRITICAL NODES). *Given an integer program $\mathbb{P}(\mathcal{P}, \mathcal{V}, \mathcal{C})$ over a (merged) set of partitions $\mathcal{P}$ as constructed using Algorithm* CPALGO *and Algorithm 1, consider the program $\mathbb{P}'(\mathcal{P}, \mathcal{V}, \mathcal{C}')$, where $\mathcal{C}'$ is obtained from $\mathcal{C}$ by removing all zoom consistency and visibility constraints (Equation 4) that are not on critical nodes. We then have that $\mathbb{P} \equiv \mathbb{P}'$, i.e., every solution to $\mathbb{P}$ ($\mathbb{P}'$, resp.) is also a solution to $\mathbb{P}'$ ($\mathbb{P}$, resp.).*

### Bounded Cover of Regions

While Definition 2.2 defines a region by any subset $S \subseteq \mathcal{N}^{\mathcal{Z}}$, we can typically define regions by a *bounded cover*, i.e., by a set of cover nodes $C \subseteq \mathcal{N}$, where $C$ is a set of (possibly internal) nodes of the tree and $|C| \leq M$ for some fixed constant $M$. Intuitively, the set $S$ corresponding to all level-$\mathcal{Z}$ nodes is the set of all descendants of $C$. While using a bounded cover may require approximation of a very complex region and thereby compromise optimality, it improves efficiency. In our implementation we use $M = 8$, which is what is also used in our commercial offering of Fusion Tables [14]. The bounded cover of size $M$ for every region imposes a bound on the number of critical nodes.

LEMMA 3.4. *Given a geoset $G = \{R_1, \ldots, R_n\}$ with bounded covers of size $M$ over a spatial tree $T(\mathcal{Z}, \mathcal{N})$, the number of critical nodes in our integer programming formulation $\mathbb{P}$ is at most $nM\mathcal{Z}$.*

### Summary

The optimizations we described above yield the main result of this section: an integer program of size linear in the input.

THEOREM 3.2. *Given a geoset $G = \{R_1, \ldots, R_n\}$ with a bounded cover of size $M$ over a spatial tree $T(\mathcal{Z}, \mathcal{N})$, and a maximum bound $K \in \mathbb{N}$ on the number of visible records in any cell, there exists an equivalent integer program $\mathbb{P}(\mathcal{P}, \mathcal{V}, \mathcal{C})$ constructed from Algorithms 1 and* CPALGO *with constraints on critical nodes such that $|\mathcal{V}| = \mathcal{Z}|\mathcal{P}| = \mathcal{O}(n\mathcal{Z})$ and $|\mathcal{C}| = \mathcal{O}(nM\mathcal{Z})$.*

### 3.1.3 Modeling objectives in the integer program

We now describe how objective functions are specified. The objective is described by a function over the set of variables $\mathcal{V}$.

To maximize the number of records visible across all cells, the following objective $\mathcal{F}_{max}$ represents the aggregate number of records (counting each record $x$ times if it is visible in $x$ cells):

$$\mathcal{F}_{max} = \sum_{c_j^z \in \mathcal{N}} \sum_{q: c_j^z \in span(P_q)} \sum_{z^* \leq z} v_q^{z^*} \tag{5}$$

Instead, if we wish to maximize the number of *distinct* records visible at any cell, we may use the following objective:

$$\mathcal{F}_{distinct} = \sum_{v_q^z \in \mathcal{V}} v_q^z$$

The following objective captures fairness of records: it makes the total number of records sampled from each partition as balanced as possible.

$$\mathcal{F}_{fair} = -\left( \sum_{P_q \in \mathcal{P}} V(P_q)^2 \right)^{\frac{1}{2}}$$

where $V(P_q) = \sum_{c_j^z} \sum_{z^* \leq z} v_q^{z^*}$, i.e., the total number of records visible (at some zoom level) from the partition $P_q$, aggregated over all cells. The objective above gives the $L_2$ norm of the vector with $V$ values for each partition. The fairness objective is typically best used along with another objective, e.g., $\mathcal{F}_{max} + \mathcal{F}_{fair}$. Further, in order to capture fairness within a partition, we simply treat each record in a partition uniformly, as we describe shortly.

To capture importance of records, we can create the optimization problem by subdividing each partition $P_q$ into equivalence classes based on importance of records. After this, we obtain a revised program $\mathbb{P}(\mathcal{P}', \mathcal{V}, \mathcal{C})$ and let $\mathcal{I}(P_q)$ denote the importance of each record in partition $P_q \in \mathcal{P}'$. We may then incorporate the importance into our objective as follows:

$$\mathcal{F}_{imp} = \sum_{c_j^z \in \mathcal{N}} \sum_{q: c_j^z \in span(P_q)} \sum_{z^* \leq z} \mathcal{I}(P_q) v_q^{z^*} \tag{6}$$

Other objective functions, such as combining importance and fairness can be incorporated in a similar fashion.

EXAMPLE 3.1. *Continuing with the solutions in Example 2.4 using data in Figure 3, let us also add another solution $M^4(\cdot)$ with $M^4(R5) = 3$, $M^4(R1) = 1$ and $M^4(Ri) = 4$ for all other records. Further, suppose we incorporate importance into the records and set the importance of $R2, R3$ to 10, and the importance of every other record to 1.*

*Table 1 compares each of the objective functions listed above on all these solutions. Since $M^1$ doesn't show any records, its objective value is always 0. $M^2$ shows two distinct records $R1$ and $R2$, $R1$ shown in 3 cells, and $R2$ shown*

| | $\mathcal{F}_{max}$ | $\mathcal{F}_{distinct}$ | $\mathcal{F}_{fair}$ | $\mathcal{F}_{imp}$ |
|---|---|---|---|---|
| $M^1$ | 0 | 0 | 0 | 0 |
| $M^2$ | 4 | 2 | -3.16 | 13 |
| $M^3$ | 3 | 2 | -2.24 | 12 |
| $M^4$ | 5 | 2 | -3.61 | 5 |

**Table 1: Table comparing the objective measures for various solutions in Example 3.1.**

*in one cell giving $\mathcal{F}_{max}$ and $\mathcal{F}_{distinct}$ values as 4 and 2. Since $M^2$ shows records in 3, 1, 0, and 0 cells from the partitions $\{R1\}$, $\{R2, R3\}$, $\{R4\}$, $\{R5\}$ respectively, $\mathcal{F}_{fair}(M^2) = 20$, and using the importance of R2, we get $\mathcal{F}_{imp} = 13$. Similarly, we compute the objective values for other solutions. Note that $M^4$ is the best based on maximality, and $M^2$ is the best based on importance. Note that our objective of combining fairness, i.e., using $\mathcal{F}_{max} + \mathcal{F}_{fair}$, gives $M^4$ as the best solution. Finally, these solutions aren't distinguished based on the distinct measure.*

## 3.2 Relaxing the integer constraints

In addition to the integer program described above, we also consider a relaxed program $\mathbb{P}^r$ that is obtained by eliminating the integer constraints (Equation (3)) on $v_q^z$'s. The relaxed program $\mathbb{P}^r$ is typically much more efficient to solve since integer programs often require exponential-time, and can be converted to an approximate solution. We then perform sampling just as above, except, we sample $\lfloor v_q^z \rfloor$ regions. The resulting solution still satisfies all constraints, but may be sub-optimal. Also, from the solution to $\mathbb{P}^r$, we may compute the objective values $\mathcal{F}^{ub}(\mathbb{P}^r)$, and the true objective value obtained after rounding down as above, denoted $\mathcal{F}(\mathbb{P}^r)$. It can be seen easily that:

$$\mathcal{F}(\mathbb{P}^r) \leq \mathcal{F}(\mathbb{P}) \leq \mathcal{F}^{ub}(\mathbb{P}^r)$$

In other words, the solution to $\mathbb{P}^r$ after rounding down gives the obtained value of the objective, and without rounding down gives us an upper bound on what the integer programming formulation can achieve. This allows us to accurately compute potential loss in the objective value due to the relaxation. Using this upper bound, in our experiments in Section 6, we show that in practice $\mathbb{P}^r$ gives the optimal solution in all real datasets.

# 4. MAXIMALITY

We now consider the thinning problem for a geoset $G = \{R_1, \ldots, R_n\}$, with the specific objective of maximizing the number of records shown, which is the objective pursued by Fusion Tables [14].[1]

## 4.1 Strong and weak maximality

Maximally can be defined as follows.

Definition 4.1 (Strong Maximality). *A solution $M : \{1, \ldots, n\} \to \{1, \ldots, \mathcal{Z}, \mathcal{Z}+1\}$ to thinning for a geoset $G = \{R_1, \ldots, R_n\}$ over a spatial tree $T(\mathcal{Z}, \mathcal{N})$, and a maximum bound $K \in \mathbb{N}$ on the number of visible records in any cell is said to be* strongly maximal *if there does not exist a different solution $M'$ to the same thinning problem such that*

- $\forall c \in \mathcal{N} : |Vis_M(G, T, c)| \leq |Vis_{M'}(G, T, c)|$
- $\exists c \in \mathcal{N} : |Vis_M(G, T, c)| < |Vis_{M'}(G, T, c)|$

The strong maximality condition above ensures that as many records as possible are visible at any cell. We note that the objective function $\mathcal{F}_{max}$ from Section 2.2.2 ensures strong maximality (but strong maximality doesn't ensure optimality in terms of $\mathcal{F}_{max}$).

Example 4.1. *Recall the data from Figure 3, and consider solutions $M^1, M^2, M^3$ and $M^4$ from Example 2.4 and 3.1. It can be seen that $M^4$ is a strongly maximal solution: All non-empty cells show exactly one region, and since $K = 1$, this is a strongly maximal solution. Note that $M^2$ (and hence $M^1$ and $M^3$) from Example 2.4 are not strongly maximal, since $c_3^3$ does not show any record and $M^4$ above shows same number of records as $M^2$ in all other cells, in addition to $c_3^3$.*

Unfortunately, as the following theorem states, finding a strongly maximal solution to the thinning problem is intractable in general. (The proof is by a reduction from the NP-hard Exact Set Cover problem [13].)

Theorem 4.1 (Intractability of Strong Maximality). *Given a geoset $G = \{R_1, \ldots, R_n\}$ over a spatial tree $T(\mathcal{Z}, \mathcal{N})$, and a maximum bound $K \in \mathbb{N}$, finding a strongly maximal solution to the thinning problem is NP-hard in $n$.*

Fortunately, there is a weaker notion of maximality that does admit efficient solutions. Weak maximality, defined below, ensures that no individual record can be made visible at a coarser zoom level:

Definition 4.2 (Weak Maximality). *A solution $M : \{1, \ldots, n\} \to \{1, \ldots, \mathcal{Z}, \mathcal{Z}+1\}$ to thinning for a geoset $G = \{R_1, \ldots, R_n\}$ over a spatial tree $T(\mathcal{Z}, \mathcal{N})$, and a maximum bound $K \in \mathbb{N}$ on the number of visible records in any cell is said to be* weakly maximal *if for any $M' : \{1, \ldots, n\} \to \{1, \ldots, \mathcal{Z}, \mathcal{Z}+1\}$ obtained by modifying $M$ for a single $i \in \{1, \ldots, n\}$ and setting $M'(i) < M(i)$, $M'$ is not a thinning solution.*

Example 4.2. *Continuing with Example 4.1, we can see that $M^2$ (defined in Example 2.4) and $M^4$ are weakly maximal solutions: we can see that reducing the $M^2$ value for any region violates the visibility bound of $K = 1$. For instance, setting $M^2(R5) = 3$ shows two records in $c_4^3$. Further, $M^3$ from Example 2.4 is not weakly maximal, since $M^2$ is a solution obtained by reducing the min-level of $R1$ in $M^3$.*

The following lemma expresses the connection between strong, weak maximality, and optimality under $\mathcal{F}_{max}$ from Section 2.2.2.

Lemma 4.1. *Consider a thinning solution $M : \{1, \ldots, n\} \to \{1, \ldots, \mathcal{Z}, \mathcal{Z}+1\}$ to for a geoset $G = \{R_1, \ldots, R_n\}$ over a spatial tree $T(\mathcal{Z}, \mathcal{N})$, and a maximum bound $K \in \mathbb{N}$ on the number of visible records in any cell.*

- *If $M$ is optimal under $\mathcal{F}_{max}$, then $M$ is strongly-maximal.*
- *If $M$ is strongly-maximal, then $M$ is weakly-maximal.*
- *If $M$ is weakly-maximal and $G$ only consists of point records, then $M$ is strongly-maximal.*

[1] Our algorithms will satisfy restricted fairness, but maximality is the primary subject of this section.

**Algorithm 2** DFS algorithm for thinning.

**Input:** Geoset $G = \{R_1, \ldots, R_n\}$ over spatial tree $T(\mathcal{Z}, \mathcal{N})$, visibility bound $K \in \mathbb{N}$.
**Output:** Min-level function $M : \{1, \ldots, n\} \to \{1, \ldots, \mathcal{Z}+1\}$.
Initialize $\forall i \in \{1, \ldots, n\} : M(i) = \mathcal{Z} + 1$.
Initialize *Stack* $S$ with entry $(c_1^0, G)$.
// Iterate over all stack entries (DFS traversal of $T$)
**while** $S \neq \emptyset$ **do**
  Obtain top entry $(c_j^z, g \subseteq G)$ from $S$.
  Compute $Vis_M(g, T, c_j^z) = \{R_i \in g | (c_j^z \in span(R_i)) \&\& (M(i) \leq z)\}$; let $VCount = |Vis_M(g, T, c_j^z)|$.
  // Sample more records if this cell is not filled up
  **if** $VCount < K$ **then**
    Let $InVis = g \setminus Vis_M(g, T, c_j^z)$.
    // Sample up to $SCount = min\{(K - VCount), |InVis|\}$ records from $InVis$.
    **for** $R_i \in InVis$ (// in random order) **do**
      // Sampling $R_i$ shouldn't violate any visibility
      Initialize $sample \leftarrow true$
      **for** $c^z \in span(R_i)$ **do**
        **if** $Vis_M(G, T, c^z) \geq K$ **then**
          $sample = false$
      **if** sample **then**
        Set $M(R_i) = z$.
  **if** $z < \mathcal{Z}$ **then**
    // Create entries to add to the stack
    **for** $R_i \in g$ **do**
      Add $R_i$ to each child cell set $g_j$ corresponding $c_j^{z+1}$ for the children cells $R_i$ spans.
      Add all created $(c_j^{z+1}, g_j)$ entries to $S$.
**Return** $M$.

## 4.2 DFS thinning algorithm

The most natural baseline solution to the thinning problem would be to traverse the spatial tree level-by-level, in breadth-first order, and assign as many records as allowed. Instead, we describe a depth-first search algorithm (Algorithm 2) that is exponentially more efficient, due to significantly reduced memory requirements. The main idea of the algorithm is to note that to compute the set of visible records at a particular node $c_j^z$ in the spatial tree, we only need to know the set of all visible records in all ancestor cells of $c_j^z$; i.e., we need to know the set of all records from $\{R_i | c_j^z \in span(R_i)\}$ whose min-level have already been set to a value at most $z$. Consequently, we only need to maintain at most $4\mathcal{Z}$ cells in the DFS stack.

Algorithm 2 proceeds by assigning every record to the root cell of the spatial tree, and adding this cell to the DFS stack. While the stack is not empty, the algorithm picks the topmost cell $c$ from the stack and all records that span $c$. The required number of records are sampled from $c$ so as to obtain up to $K$ visible records; then all the records in $c$ are assigned to $c$'s 4 children (unless $c$ is at level $\mathcal{Z}$), and these are added into the stack. While sampling up to $K$ visible records, we ensure that no sampled record $R$ increases the visibility count of a different cell at the same zoom level to more than $K$; to ensure this, we maintain a map from cells in the tree (spanned by some region) to their visibility count (we use $Vis$ to denote this count).

The theorem below summarizes properties of Algorithm 2.

THEOREM 4.2. *Given a geoset $G = \{R_1, \ldots, R_n\}$ over spatial tree $T(\mathcal{Z}, \mathcal{N})$, and visibility bound $K \in \mathbb{N}$, Algorithm 2 returns:*

1. *A weakly maximal thinning solution.*
2. *A strongly maximal thinning solution if $G$ only consists of records with point records.*

**Algorithm 3** A randomized thinning algorithm for geosets of point records.

**Input:** Geoset $G = \{R_1, \ldots, R_n\}$ of point records over spatial tree $T(\mathcal{Z}, \mathcal{N})$, spatial index $\mathcal{I}$ visibility bound $K \in \mathbb{N}$.
**Output:** Min-level function $M : \{1, \ldots, n\} \to \{1, \ldots, \mathcal{Z}+1\}$.
Initialize $\forall i \in \{1, \ldots, n\} : M(i) = \mathcal{Z} + 1$.
**for** $i = 1 \ldots n$ **do**
  Set $priority(R_i) = Rand()$.
**for** Non-empty cells $c_j^z \in \mathcal{I}$ **do**
  $K' = \min\{|\mathcal{I}(c_j^z)|, K\}$
  **for** $R_i \in$ top-$K'(\mathcal{I}(c_j^z))$ **do**
    **if** $M(i) > z$ **then**
      Set $M(i) = z$
**Return** $M$.

*The worst-case time complexity of the algorithm is $O(n\mathcal{Z})$ and its memory utilization is $O(4\mathcal{Z})$.*

The following simple example illustrates a scenario where Algorithm 2 does not return a strongly maximal solution.

EXAMPLE 4.3. *Continuing with the data from Figure 3, suppose at $z = 1$ we randomly pick $R1$, and then at $z = 3$, we sample $R2$ from $c_4^3$. We would then end up in the solution $M^2$, which is weakly maximal but not strongly maximal (as already described in Example 4.2).*

# 5. POINT ONLY DATASETS

We present a randomized thinning algorithm for a geoset $G = \{R_1, \ldots, R_n\}$ consisting of only point records over spatial tree $T(\mathcal{Z}, \mathcal{N})$.

The main idea used in the algorithm is to exploit the fact that no point spans multiple cells at the same zoom level: i.e., for any point record $R$ over spatial tree $T(\mathcal{Z}, \mathcal{N})$, if $c_{j_1}^z, c_{j_2}^z \in span(R)$ then $j_1 = j_2$. Therefore, we can obtain a global total ordering of all points in the geoset $G$, and for any cell simply pick the top $K$ points from this global ordering and make them visible.

The algorithm (see Algorithm 5) first assigns a real number for every point independently and uniformly at random (we assume a function *Rand* that generates a random real number in $[0, 1]$; this random number determines the total ordering among all points). Then for every record we assign the coarsest zoom level at which it is among the top $K$ points based on the total order.

To perform this assignment, we pre-construct a spatial index $\mathcal{I} : \mathcal{N} \to 2^G$, which returns the set of all records spanning a particular cell in the spatial tree $T$. That is, $\mathcal{I}(c) = \{R_i | c \in span(R_i)\}$, and the set of records are returned in order of their random number. This spatial index can be built in standard fashion (such as [19, 16]) in $O(n \log n)$ with one scan of the entire dataset. Assignment of the zoom level then requires one index scan.

THEOREM 5.1 (RANDOMIZED ALGORITHMS FOR POINTS). *Given a geoset $G = \{R_1, \ldots, R_n\}$ of point records over spatial tree $T(\mathcal{Z}, \mathcal{N})$, spatial index $\mathcal{I}$, and visibility bound $K \in \mathbb{N}$, Algorithm 5 returns a strongly maximal solution to the thinning problem with an offline computation time $O(n\mathcal{Z})$, and constant (independent of the number of points) memory requirement.*

Furthermore, Algorithm 5 also has several other properties that make it especially attractive in practice.

1. The second step of assigning $M(i)$ for each $i = 1..n$ doesn't necessarily need to be performed offline. Whenever an application is rendering the set of points on a map, it can retrieve the set of points in sorted order based on the random number, and simply display the first $K$ points it obtains.
2. If we have pre-existing importance among records, the algorithm can use them to dictate the priority assigned, instead of using a random number. For example, in a restaurants dataset, if we want to show more popular restaurants, we can assign the priority based on the star-ratings of each restaurant (breaking ties randomly).
3. The algorithm can be extended easily to large geosets that don't necessarily fit in memory and are partitioned across multiple machines. The assignment of a random number on each point happens independently and uniformly at random. Thereafter, each partition picks the top-$K$ points for any cell based on the priority, and the overall top-$K$ are obtained by merging the top-$K$ results from each individual partition.

# 6. EXPERIMENTS

This section presents a detailed experimental evaluation of our algorithms. After presenting our datasets and experimental setup in Section 6.1, we present the following main experimental findings:

1. In Section 6.2, we show that the optimization program minimization techniques from Section 3.1.2 usually reduces the size of the problem by more than two orders of magnitude.
2. In Section 6.3, we show that in all seven of our datasets, the integer relaxation (Section 3.2) doesn't affect optimality as compared to the integer formulation.
3. Section 6.4 looks at scalability. The optimization program without minimizing program size scales only until around thousands of records, while after program-size minimization it scales to hundreds of thousands of records. A baseline tree-traversal algorithm scales to around ten million records, while our DFS traversal algorithm scales to around 20 million records, after which they get bottlenecked by memory.
4. In Section 6.5, we study objectives other than maximality, i.e., fairness and importance. First we show that for the importance-based objective of $\mathcal{F}_{imp}$, the optimization program gives the best solution (as expected), but $DFS$ also gives a close solution. Further, we show that as skew in the importance increases, the value of incorporating importance into the objective also increases. Then we present a qualitative study of how fairness ensured by the optimization program's objective improves the thinning solution by sampling records from regions in a roughly uniform fashion.
5. Finally, Section 6.6 gives a breakup of the optimization solution, showing that most of the time is spent in building and solving the problem, while sampling after that is negligible.

The main takeaways from the experiments are: (1) When we care about maximality only, then the DFS algorithm presents a high-quality and efficient solution; (2) For all other objectives, the optimization program along with the problem minimization techniques from this paper present a practical solution.

## 6.1 Experimental setup

We used the following real-world datasets containing points, lines and polygons, and their sizes varying from a few thousand records to more than 60 million. All the following datasets are real data uploaded to our commercially-used Fusion Tables system [14].

| Name | Type | # records | # points |
|---|---|---|---|
| Theft | point | 2,526 | 2,526 |
| Flu | point | 6,776 | 6,776 |
| U.S. county | polygon | 3,241 | 32,046 |
| Hiking Trails | line | 5,211 | 399,387 |
| Ecoregion | polygon | 14,458 | 3,933,974 |
| Trajectory | point | 716,133 | 716,133 |
| U.S. Parcel | point | 61,924,397 | 61,924,397 |

These datasets describe: (1) the locations of motor vehicle thefts in Colier County, (2) the pharmacies and clinic locations in U.S. offering Flu vaccines, (3) the polygons of all counties in the U.S., (4) popular hiking and biking trails in the world, (5) the set of eco-regions in the world [22], (6) trajectories of individuals of a location-based social networking service, (7) the set of all housing parcels in the U.S.

We implemented and evaluated the following algorithms. The first three are based on the integer programming solution, the next three are DFS and its variations, and the final one is the randomized algorithm for points.

- $Opt_{naive}$ is the integer program but without our proposed optimizations from Section 3.1.2. Each record forms a single partition.
- $Opt_{max}$ is the algorithm described in Section 3 with objective $\mathcal{F}_{max}$ in Equation (5).
- $Opt_{imp}$ is the algorithm described in Section 3 with objective $\mathcal{F}_{imp}$ in Equation (6). Importance of a record is a number between 0 and 1; we experimented with importance chosen uniformly at random for each record, as well as using a zipfian distribution. We discretize the range and create equivalence classes by subdividing it into 10 buckets: (0, 0.1], (0.1, 0.2], ... (0.9, 1).
- $DFS$ implements Algorithm 2, i.e., a depth-first search.
- $BFS$ is a baseline algorithm that is similar to Algorithm 2, but instead traverses the spatial tree in a level-by-level fashion, starting from the root, then sampling for every node in the root's children, and so on.
- $DFS_{imp}$ is the same as $DFS$, but performs weighted sampling based on the record importance.
- *Rand* is Algorithm 5, which works for point datasets.

We use $Opt_{naive}$ only to demonstrate how well the optimization framework can scale without the minimization technique. Since *Rand* only needs to assign random numbers to records and does not involve any runtime thinning overhead, we do not include figures from *Rand*. *Rand* consumes only a constant memory and scales well to arbitrarily large datasets.

All algorithms were implemented in Java 1.6. We used Apache Simplex Solver[1] for our linear optimization. The solver is a linear programming (LP) solver. We relaxed the integer constraints as proposed in Section 3.2 and rounded down solutions from the solver. We ran all experiments on a desktop PC running Linux kernel 2.6.32 on a 2.67 GHz Intel quad core processor with 12 GB of main memory. All exper-

iments were performed in-memory with a default memory of 1GB except the one for scalability where we used 4GB. The visibility bound $K$ was set to 500. For most figures, we only show four datasets, since the values (e.g., $\mathcal{F}_{imp}$) are at a different scale and don't fit in the plot; however, for our scalability experiments we present results on the largest U.S. parcel dataset.

## 6.2 Benefit of minimizing program size

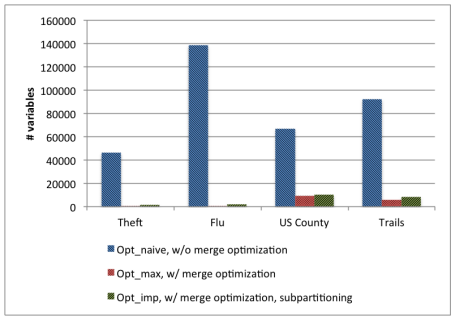



**Figure 4: Impact of Merging Partitions**

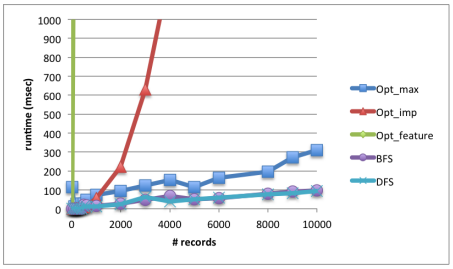
We show effectiveness of the program size minimization techniques in Section 3.1.2. Figure 4 shows the number of variables input to the solver. The first bar of each dataset is the number of variables before applying the optimization techniques in Section 3.1.2. The second bar is the reduced number of variables after merging partitions and considering critical nodes. In general there is more than a two order of magnitude reduction in the number of variables. For Flu, there were originally 138,726 variables, but after minimizing the program size, the number was reduced to 229. The reduction in the number of constraints was similar. The number of variables increases in $Opt_{imp}$ because of its subpartitioning based on equivalence classes on importance. Without the proposed techniques for program size minimization, it is virtually impossible to efficiently solve an optimization problem of this scale.

## 6.3 Integer relaxation

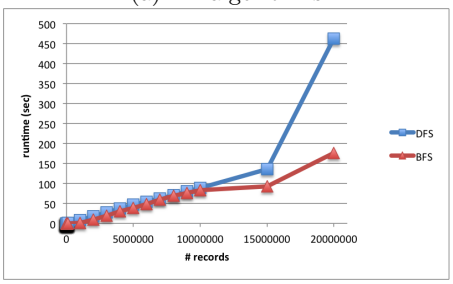
We compared our integer program solution with the relaxed solution (Section 3.2). Although the relaxed solution can theoretically be sub-optimal, in all 7 datasets we observed identical solutions (i.e., relaxed solutions had integral variable values), due to largely non-conflicting spatial distributions of records. This shows that employing the relaxed solution does not affect optimality (significantly).

## 6.4 Scalability

We study scalability using the US Parcel dataset, which is our largest dataset. Figure 5 plots runtime versus the number of records. To properly show the scale, Figure 5(a) plots a small data size range (up to 100,000 records), and Figure 5(b) plots a larger data size range (up to 20 million records) showing BFS and DFS. We stop plotting an algorithm if it takes more than 10 minutes or needed more than 4G of memory. It is obvious that $Opt_{naive}$ is not scalable at all. It shows very sharp increase in runtime from the beginning and cannot even handle thousands of records. $Opt_{max}$ performs well until hundreds of thousands of records, but after that the problem solving time becomes the bottleneck.




(a) All algorithms




(b) BFS & DFS

**Figure 5: Scalability**

$Opt_{imp}$ generates more number of variables and constraints, and thus is slower than $Opt_{max}$.

$BFS$ and $DFS$ outperform the optimization-based techniques by a large margin. The performance of $BFS$ starts to degrade at around ten million records. This is largely due to the cost of memory management. At each stage, the algorithm holds records corresponding to all nodes under processing, which can consume a large amount of memory. However, in $DFS$, there are at most $\mathcal{Z}$ nodes at any given time, so it is much more efficient. We observe that $DFS$ scales fairly well up to 20 million records.

However, even $DFS$ does not scale up above tens of millions of records due to its memory requirement. For point datasets, *Rand* only consumes a constant amount memory and can handle arbitrarily large datasets, including the Parcel dataset. To handle large polygon datasets, we are exploring algorithms that are distributed over multiple machines. The details are left for future work.

## 6.5 Importance and fairness objectives

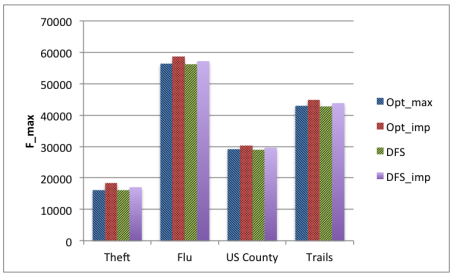



**Figure 6: Objective Based on Uniform Importance**

First we consider optimality in datasets with importance. Figure 6 shows $\mathcal{F}_{imp}$ values of various algorithms. By optimizing for $\mathcal{F}_{imp}$, we can see $Opt_{imp}$ achieves the highest objective value for all data sets. We note that the objective values of $DFS$ and $DFS_{imp}$ are very close to that of $Opt_{max}$, with $DFS_{imp}$ being better than $DFS$. Further, as shown

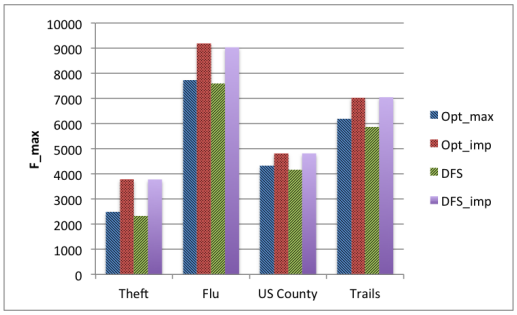



**Figure 7: Objective Based on Zipfian Importance**

in Figure 7, using a zipfian distribution for importance enhances the gap between importance-based algorithms versus the importance-agnostic ones; in general, the more skew there is in data, the more important it is to consider importance in the objective. And we shall show shortly that the DFS solutions are very efficient; hence, we infer that for maximality, the DFS solutions is most appropriate.

We next present the impact of considering fairness. We qualitatively compare the results of two different objective functions: $\mathcal{F}_{max}$ and $\mathcal{F}_{imp}$. Figure 8(a) shows the result from maximizing $\mathcal{F}_{max}$. Notice that the artifact of partitions are visible (as rectangular holes). This is because $\mathcal{F}_{max}$ only tries to maximize the sum, and may assign a large value to one variable as long as the assignment does not hurt the goal. In the example, the solver assigned 0 to variables corresponding to empty holes assigning high values to others. While $\mathcal{F}_{max}$ only cares about maximality, $\mathcal{F}_{imp}$ considers importance. As we assign importance uniformly at random and subdivide each partition according to the importance, the solver is not likely to choose everything from one partition and nothing from the other. Figure 8(b) depicts the result from $\mathcal{F}_{imp}$ with random importance. We can see points are much more naturally distributed without seeing artifacts of partitioning.

We note that using $\mathcal{F}_{imp}$ is one of many possible ways to consider fairness. The $L_2$ norm or adding a term for minimizing deviation from the mean are other examples, some of which would require a more powerful solver such as CPLEX [4].

## 6.6 Optimization runtime

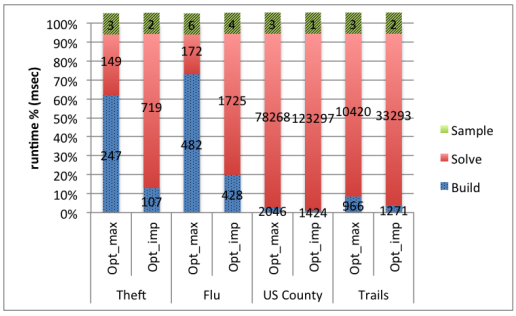



**Figure 9: Breakup of Runtime**

Figure 9 presents the break-down of runtime of each of the optimization programs. For $Opt_{max}$ and $Opt_{imp}$, we see a large fraction of the runtime is spent in building and solving the optimization program. $Opt_{imp}$ is the slowest due to increased number of variables from subpartitioning. For larger datasets, the problem solving is the dominating part. A more powerful solver, such as CPLEX, will reduce the runtime greatly.

# 7. RELATED WORK

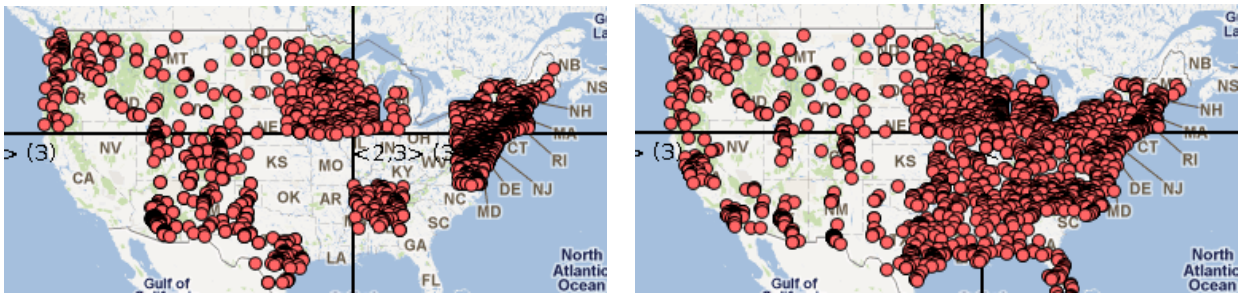
While map visualizations of geographical data are used in multiple commercial systems such as Google Maps [6] and MapQuest [7], we believe that ours is the first paper to formally introduce and study the thinning problem, which is a critical component in some of these systems. The closest body of related research work is that of *cartographic generalization* [12, 25].

Cartographic generalization deals with selection and transformation of geographic features on a map so that certain visual characteristics are preserved at different map scales [12, 28, 29]. This work generally involves domain expertise in performing transformations while minimizing loss of detail (e.g., merging nearby roads, aggregating houses into blocks, and blocks into neighborhoods), and is a notoriously difficult problem [12]. Our work can be used to complement cartographic generalization in two ways. First, it can filter out a subset of features to input into the generalization process, and second, it can select a subset of the transformed features to render on a map. For example, you could assign importance to road segments in a road network, use our method to select the most important segments in each region, and then generalize those roads through an expensive cartographic generalization process. A process related to thinning is spatial clustering [18], which can be used to summarize a map by merging spatially close records into clusters. A major difference in our work is imposing spatial constraints in the actual sampling of records.

Multiple studies have shown that clutter in visual representation of data can have negative impact in user experience [24, 30]. The *principle of equal information density* from the cartographic literature states that the number of objects per display unit should be constant [12]. The proposed framework can be thought of as an automated way to achieve similar goals with constraints. DataSplash is a system that helps users construct interactive visualizations with constant information density by giving users feedback about the density of visualizations [30]. However, the system does not automatically select objects or force constraints.

The vast literature on top-$K$ query answering in databases (refer to [21] for a survey) is conceptually similar since even in thinning we effectively want to show a small set of features, as in top-$K$. However, work on top-$K$ generally assumes that the ranking of tuples is based on a pre-defined (or at least independently assigned) *score*. However, the main challenge in thinning is that of picking the right set of features in a holistic fashion (thereby, assigning a "score" per region per zoom level, based on the objective function and spatial constraints). Therefore, the techniques from top-$K$ are not applicable in our setting.

Spatial data has been studied extensively in the database community as well. However, the main focus has been on data structures, e.g. [16, 27], query processing, e.g. [15, 20], spatial data mining, e.g. [17] and scalability, e.g. [23]; these are all largely orthogonal to our contributions. The spatial index in Section 2 can be implemented with various data structures studied, e.g. [16, 19].




(a) Flu with $F_{max}$

(b) Flu with $F_{imp}$

**Figure 8: Results with Different Objective Functions**

Sampling is a widely studied technique that is used in many areas [10]. We note that our primary goal is to decide the number of records to sample, while the actual sampling is performed in a simple uniformly random process.

Finally, a large body of work has addressed the problem of efficiently solving optimization problems. We used Apache Simplex Solver [1] for ease of integration with our system. Other powerful packages, such as CPLEX [4] also may be used. The idea of converting an integer program into a relaxed (non-integer) formulation in Section 3.2 is a standard trick applied in optimization theory in order to improve efficiency (by potentially compromising on optimality) [9].

# 8. CONCLUSIONS

We introduced and studied the thinning problem of efficiently sampling regions from a geographical dataset for visualization on a map. The main challenges in the thinning problem are effectively balancing spatial constraints imposed by commercial maps systems (such as zoom consistency, visibility bound, and adjacency) with objective criteria (such as maximality, fairness, and record importance), while scaling to tens of millions of records. We introduced an optimization framework that captures all constraints, and any general objective function, and showed how to perform several improvements to the base model to reduce the problem to linear size. As our next contribution, we considered the objective of maximality and showed intractability results, and more efficient algorithms. We then considered the common case of points and showed an effective randomized algorithm. Finally, we presented detailed experimental results on real datasets in our commercial Fusion Tables system [14], demonstrating the effectiveness of our techniques.

# 9. REFERENCES


[1] Apache simplex solver. http://commons.apache.org/math/.

[2] Arcgis. http://www.esri.com/software/arcgis/index.html.

[3] Cartodb. http://cartodb.com.

[4] Cplex. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/.

[5] Geocommons. `http://geocommons.com/`.

[6] Google maps. http://maps.google.com.

[7] Mapquest. http://mapquest.com.

[8] Oracle spatial. `http://www.oracle.com/us/products/database/options/spatial/index.html`.

[9] S. Agmon. The relaxation method for linear inequalities. *Canadian Journal of Mathematics*, 5(3):388–414, 1954.

[10] W. G. Cochran. *Sampling Techniques, 3rd Edition*. John Wiley, 1977.

[11] S. Cohen, C. Li, J. Yang, and C. Yu. Computational journalism: A call to arms to database researchers. In *CIDR*, pages 148 – 151, 2011.

[12] A. U. Frank and S. Timpf. Multiple representations for cartographic objects in a multi-scale tree - an intelligent graphical zoom. *Computers and Graphics*, 18(6):823 – 829, 1994.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[14] H. Gonzalez, A. Y. Halevy, C. S. Jensen, A. Langen, J. Madhavan, R. Shapley, W. Shen, and J. Goldberg-Kidon. Google fusion tables: web-centered data management and collaboration. In *SIGMOD Conference*, 2010. http://www.google.com/fusiontables.

[15] S. Grumbach, P. Rigaux, and L. Segoufin. The dedale system for complex spatial queries. In *SIGMOD Conference*, 1998.

[16] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD Conference*, pages 47–57, 1984.

[17] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000.

[18] J. Han, M. Kamber, and A. K. H. Tung. Spatial clustering methods in data mining: A survey. *Geographic Data Mining and Knowledge Discovery*, pages 1 – 29, 2001.

[19] D. Hilbert. Uber die stetige abbildung einer linie auf ein flachenstuck. *Math. Ann.*, 38:459–460, 1891.

[20] G. R. Hjaltason and H. Samet. Incremental distance join algorithms for spatial databases. In *SIGMOD Conference*, pages 237–248, 1998.

[21] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4):11–58, 2008.

[22] D. M. Olson, E. Dinerstein, E. Wikramanayake, N. Burgess, G. Powell, E. Underwood, J. D'amico, I. Itoua, H. Strand, J. Morrison, C. Loucks, T. Allnutt, T. Ricketts, Y. Kura, J. Lamoreux, W.W.Wettengel, P. Hedao, and K. Kassem. Terrestrial ecoregions of the world: A new map of life on earth. *BioScience*, 51:933–938, 2001.

[23] J. Patel, J. Yu, N. Kabra, K. Tufte, B. Nag, J. Burger, N. Hall, K. Ramasamy, R. Lueder, C. Ellmann, J. Kupsch, S. Guo, J. Larson, D. Dewitt, and J. Naughton. Building a scalable geo-spatial dbms: Technology, implementation, and evaluation. In *SIGMOD Conference*, pages 336–347, 1997.

[24] R. Phillips and L. Noyes. An investigation of visual clutter in the topographic base of a geological map. *Cartographic Journal*, 19(2):122 – 131, 1982.

[25] E. Puppo and G. Dettori. Towards a formal model for multiresolution spatial maps. In *International Simposium on Large Spatial Database*, pages 152–169, 1995.

[26] H. Sagan. *Space-Filling Curves*. Springer-Verlag, 1994.

[27] H. Samet. *The design and analysis of spatial data structures*. Addison-Wesley Longman Publishing Co., Inc., 1990.

[28] K. Shea and R. Mcmaster. Cartographic generalization in a digital environment: When and how to generalize. *AutoCarto*, 9:56–67, 1989.

[29] M. J. Ware, C. B. Jones, and N. Thomas. Automated map generalization with multiple operators: a simulated annealing approach. *International Journal of Geographical Information Science*, 17(8):743 – 769, 2003.

[30] A. Woodruff, J. Landay, and M. Stonebraker. Constant information density in zoomable interfaces. In *AVI*, pages 57–65, 1998.