Consistent Thinning of Large Geographical Data for Map Visualization

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

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. Fourth, we show that *contiguous* regions are tractable for a general version of maximality, for which arbitrary regions are intractable. Fifth, we examine the structure of our integer programming formulation and show that for point datasets, our program is integral. Finally, we have implemented all techniques from this paper in Google Maps [Google 2005] visualizations of Fusion Tables [Gonzalez et al. 2010], 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

Additional Key Words and Phrases: geographical databases, spatial sampling, maps, data visualization, indexing, query processing

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 [Esri 2012],[Vizzuality 2012],[GeoIQ 2012], [Oracle 2007], [Gonzalez et al. 2010]. 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 [Cohen et al. 2011].

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.

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

Fig. 1. Violation of Zoom Consistency

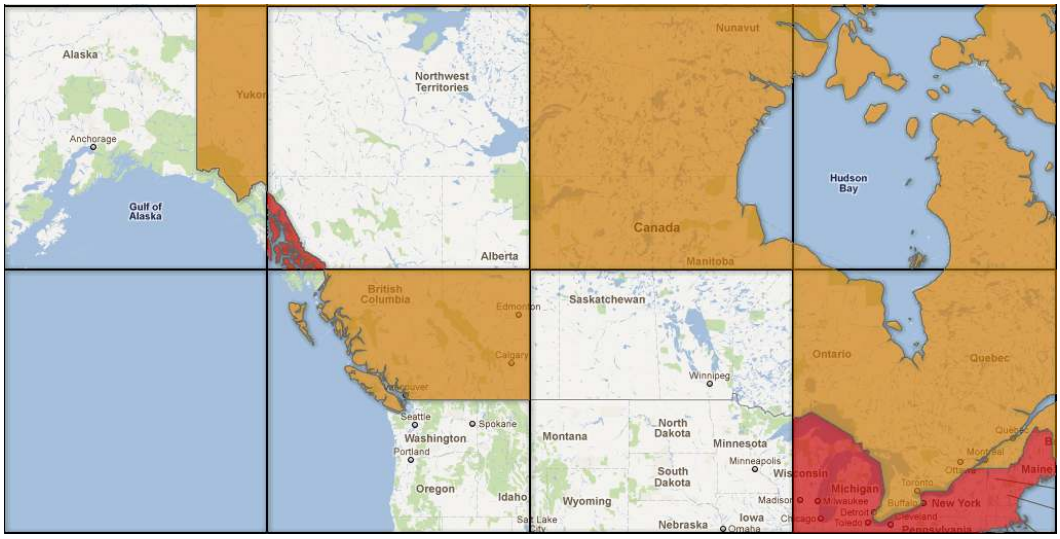


Fig. 2. Violation of Adjacency Constraint

- 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 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.

Fourth, we consider datasets that consist only of "contiguous" regions. We show that the intractability of strong maximality is due to non-contiguous regions. We present a polynomial-time algorithm for strong maximality when a dataset consists only of contiguous regions.

Fifth, we perform a detailed investigation of the structure of our integer programming formulation. We show the result that when the dataset contains only points, the programming formulation is in fact integral, leading to an efficient, exact solution.

Finally, we describe a detailed experimental evaluation of our techniques over large-scale real datasets in Google Fusion Tables [Gonzalez et al. 2010]. The experiments show that the proposed solutions efficiently select records respecting aforementioned constraints.

Section 2 discusses the related area of cartographic generalization, and presents other related work. The rest of the paper is organized as follows. Section 3 defines the thinning problem formally. Section 4 describes the integer programming solution to the thinning problem. Section 5 studies in detail maximality for arbitrary regions, and Section 6 looks at the special case of datasets with point regions. Section 7 looks at datasets with contiguous regions, and Section 8 studies the structure of the linear program, presenting an integral subclass. Experiments are presented in Section 9, and we conclude in Section 10.

2. RELATED WORK

While map visualizations of geographical data are used in multiple commercial systems such as Google Maps[1] and MapQuest[2], 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* [Frank and Timpf 1994], [Puppo and Dettori 1995].

Cartographic generalization deals with selection and transformation of geographic features on a map so that certain visual characteristics are preserved at different map scales [Frank and Timpf 1994], [Shea and Mcmaster 1989], [Ware et al. 2003]. This

[1]http://maps.google.com
[2]http://mapquest.com

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 [Frank and Timpf 1994]. 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 [Han et al. 2001], 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.

Labeling in dynamic maps is a very similar problem [Petzold et al. 2003; Been et al. 2006; Been et al. 2010], which studies the placement of labels in a dynamically generated map to avoid overlaps among labels. The number of labels is typically assumed to be too large for displaying all of them in the limited space, and an appropriate selection, and placing of labels has to be performed at an interactive speed. Although the dynamic map labeling problem is very similar to the proposed thinning problem, there are a few differences. Labels can be thought of as a special case of polygons in the thinning problem, e.g., fixed sized rectangles given a zoom level. In the thinning problem, polygons are zoom-invariant. That is, the zoom level only changes the level of details of an absolute world. Whereas labels change their relative positions depending on the zoom level. Moreover, its major constraint is on avoiding overlaps among labels, while for thinning, the major concern is on the number of objects per cell. In [Been et al. 2006], a similar optimality of showing as many labels as possible over a continuous zoom level range is defined and a greed algorithm is shown to be optimal. Very similar consistencies (e.g., zoom consistency) are proposed as well. However, unique priorities among labels are assumed and the optimality is with regard to the priority, which greatly simplifies the problem. Approximation algorithms are studied in [Been et al. 2010] and their approximation factors are provided. None of the above works performed experimental evaluation with real-world datasets.

The importance of consistency in map rendering or visualization was considered previously in other studies as well, e.g., [Tryfona and Egenhofer 1997; Dix and Ellis 2002]. Tryfona and Egnhofer proposed a systematic model for the constraints that must hold when geographic databases contain multiple views of the same geographic objects at different levels of spatial resolution [Tryfona and Egenhofer 1997]. However, the consistency is on topological relations. That is, the study concerns actual change of shapes depending on the level of details such as the transition from a region with disconnected parts to an aggregated one whereas the proposed consistencies are on selection of objects. The zoom consistency is also implicitly mentioned in [Dix and Ellis 2002]. However, the proposed solution is to *resample* the data, which could be costly. Since the resampling may slow down interaction, authors employed visual effects such as smoothing transitions between datasets to ease the problem. In contrast, the proposed solutions are able to respect the consistencies without resampling the dataset.

Dealing with Big Data is has long been acknowledged as a challenge in the visualization community, e.g., [Thomas and (Eds.) 2005]. In the ATLAS system, Chan et. al partitioned large time-series data into time units such as year, month, and day and returned a fixed number of data points based on the level of data required [Chan et al. 2008]. Piringer et. al used a multithreading architecture to support interactivity [Piringer et al. 2009]. They subdivided the visualization space into layers where data is processed separately and can be reused. Fisher et. al proposed a different approach: their system shows sufficiently trustworthy incremental samples to allow

users to make decisions without fully processing the data [Fisher et al. 2012]. Stolte et.al developed a system for describing and developing multiscale visualization that supports multiple zoom paths and both data and visualization abstractions [Stolte et al. 2003]. Data cubes are employed for data abstraction and multiple zoom paths. One of the proposed patterns, thematic maps, is applicable when visualizing geographically varying dependent measures that can be summarized at multiple levels of details (such as county or state). This scheme can be thought of as pre-defining (selecting) geographical objects based on zoom levels. Our work is more concerned about selecting which geographic items to show at different zoom levels based on visualization constraints, and therefore complements the works described above.

Multiple studies have shown that clutter in visual representation of data can have negative impact in user experience [Phillips and Noyes 1982], [Woodruff et al. 1998]. The *principle of equal information density* from the cartographic literature states that the number of objects per display unit should be constant [Frank and Timpf 1994]. 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 [Woodruff et al. 1998]. However, the system does not automatically select objects or force constraints.

The vast literature on top-K query answering in databases (refer to [Ilyas et al. 2008] 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., [Guttman 1984], [Samet 1990], query processing, e.g., [Grumbach et al. 1998], [Hjaltason and Samet 1998], spatial data mining, e.g., [Han and Kamber 2000] and scalability, e.g., [Patel et al. 1997]; these are all largely orthogonal to our contributions. The spatial index in Section 3 can be implemented with various data structures studied, e.g., [Guttman 1984], [Hilbert 1891].

Sampling is a widely studied technique that is used in many areas [Cochran 1977]. 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 for ease of integration with our system. Other powerful packages, such as CPLEX also may be used. The idea of converting an integer program into a relaxed (non-integer) formulation in Section 4.2 is a standard trick applied in optimization theory in order to improve efficiency (by potentially compromising on optimality) [Agmon 1954].

This manuscript is an extended version of the conference paper [Das Sarma et al. 2012]. Beyond the conference paper, we present here two new fundamental results on tractable subclasses of thinning: (1) Section 7 considers *contiguous regions*, with the main result that strong maximality is in PTIME for contiguous regions; (2) Section 8 considers point datasets, and proves the integrality of our linear programming formulation for point datasets. Apart from the two strong results in the new sections described above, we've also expanded the conference paper by including proof sketches for all technical results; the conference version of the paper did not contain proofs.

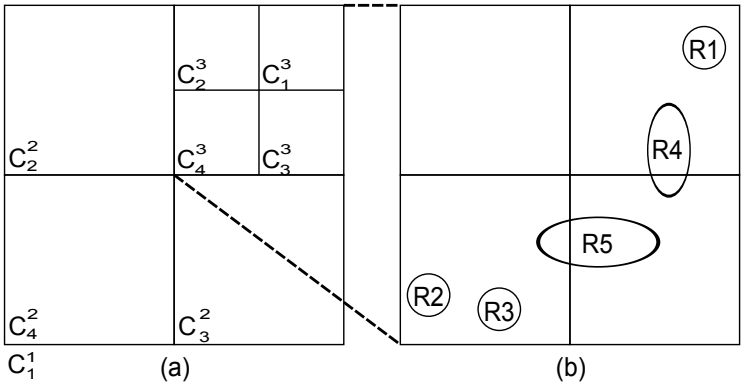


Fig. 3. Running Example: (a) Spatial tree with $\mathcal{Z} = 3$; (b) Regions shown at $z = 3$ for c_1^2.

3. DEFINITIONS

We begin by formally defining our problem setting, starting with the spatial organization of the world, defining regions and geographical datasets (Section 3.1), and then formally defining the thinning problem (Section 3.2).

3.1. 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 3.1 (*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 [Google 2005]).

Example 3.2. 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 [Sagan 1994]. 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 3.3 (*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 $|S| = 1$.

Intuitively, the shape of a region is completely specified by the set of cells at zoom level $\mathcal{Z}$ it occupies. Details finer than the cells at zoom $\mathcal{Z}$ aren't captured; for example, a region doesn't "partially" occupy a finest-granularity cell. We often refer to regions that span cells at different levels:

Definition 3.4 (*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 3.5. Figure 3(b) shows 5 regions for the cell c_1^2, showing their spans at $z = 3$ over cells $c_1^3, \ldots, c_4^3$. Although the regions are shown in "circular" shapes for clarity, these regions conform to Definition 3.3: 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 3.6 (*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.

3.2. 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 3.2.1), and then motivate some of the objectives that go into picking one among multiple thinning solutions that satisfy the constraints (Section 3.2.2).

3.2.1. 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 3.7. 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 3.1 (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 3.8. 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.

3.2.2. 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 3.9. Continuing with our data from Figure 3 and thinning solutions from Example 3.8, 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.

3.3. Outline of our solutions

In Section 4 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 5, 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 6, we study the special case of a geoset consisting of point records only. Table I provides a quick reference of the common notations we use throughout the paper.

Table I. Reference for common notations used in the paper.

Notation	Meaning
c_i^j	ith cell at zoom level j
$T(\mathcal{Z}, \mathcal{N})$	Spatial tree with $\mathcal{Z}$ levels and nodes $\mathcal{N}$
N^z	Nodes at zoom level z
$R(S)$	Region R defined by cells $S \subseteq N^{\mathcal{Z}}$
$span(R)$	Span of a region R
$Vis_M(G, T, c_j^z)$	Set of all visible records at cell c_j^z
$\mathcal{P}(G) = \{P_1, \ldots, P_l\}$	Partitioning of G into equivalence class based on region spans
v_q^z	Variable for number of records from partition P_q with min-level z

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).

4. THINNING AS INTEGER PROGRAMMING

In this section we describe an integer program that combines various objectives from Section 3.2 into the thinning problem: our program has linear constraints and we formulate multiple objective functions, some of which are non-linear. Section 4.1 describes the construction of the integer program and Section 4.2 discusses solving it.

4.1. Constructing the integer program

In this section, we shall start by discussing how to model the constraints of thinning, which are the key aspect of encoding the thinning problem as described in Section 3.2. Subsequently, we present objective functions that may be used to pick among multiple thinning solutions.

4.1.1. 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, for Constraint Program Algorithm):

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 $\mathbb{P}$ is 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:**

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

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

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

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:

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

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 4.1.
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 3.1): $\mathbb{P}$ 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 $|\mathcal{C}| = \mathcal{O}(n\mathcal{Z} + 4^{\mathcal{Z}})$.*

PROOF SKETCH. We first sketch the proof of equivalence and then prove the size of the program:

— **Equivalence of Thinning Solution and Constraint Satisfaction:** The sampling constraints are equivalent to saying that each record is either completely visible or invisible at a specific zoom level, i.e., records cannot be picked fractionally. Further, we only pick records from the input geoset G, hence the total number of records visible from a partition cannot exceed the size of the partition. The zoom consistency equation computes the total number of records visible at a zoom level z by aggregating the counts for all $z^* \leq z$, which is equivalent to the zoom consistency condition of the thinning problem. And the visibility inequality in the constraints is equivalent to the visibility bound of thinning. Finally, as described earlier, we don't need to incorporate adjacency explicitly since we have just one variable per record per zoom level.
— **Program Size:** We have one variable for each zoom level for each partition. Therefore, $|\mathcal{V}| = \mathcal{Z}|\mathcal{P}|$; since the number of partitions is at most the number of records, we have $|\mathcal{V}| = \mathcal{O}(n\mathcal{Z})$. Finally, the number of sampling constraints is $\mathcal{O}(|n\mathcal{Z}|)$, and we have one visibility constraint per cell. So, the number of constraints is $\mathcal{O}(n\mathcal{Z} + 4^{\mathcal{Z}})$.

□

4.1.2. 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).

Merging Partitions. We show how the partitions $\mathcal{P}$ generated in Section 4.1.1 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 4.2 (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*

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 $G = \{R_1, \ldots, R_n\}$ over spatial tree $T(\mathcal{Z}, \mathcal{N})$, visibility bound $K \in \mathbb{N}$; (2) Output $\mathcal{P}$, $Cover(c)$, $Touch(c)$ obtained from Algorithm CPALGO.
2: **Output:** Merged partitioning $\mathcal{P}^m$.
3: Initialize $\mathcal{P}^m = \mathcal{P}$, Stack $S = root(T)$ (i.e., the root node).
4: **while** $S \neq \emptyset$ **do**
5: Let node $c = pop(S)$.
6: // Check if c can be a valid merged partition root.
7: **if** $K \geq \sum_{P \in Touch(c)} |P|$ **then**
8: Construct merged partition $P_c = \cup_{P \in Cover(c)} P$.
9: Set $\mathcal{P}^m = (\{P_c\} \cup \mathcal{P}^m) \setminus Cover(c)$.
10: **else**
11: **if** c is not leaf **then**
12: Push each child of c into S.

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 (i.e., have the same solutions) 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., $\forall P^m \in \mathcal{P}^m \exists S \subseteq \mathcal{P} : P^m = \bigcup_{P \in S} P$*
(*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 $span(P^m) = \cup_{R_i \in P^m} span(R_i)$. Let the span of any partition or 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: $\forall P^m \in \mathcal{P}^m : |\{R_i \in G | span_{\mathcal{Z}}(R_i) \cap span_{\mathcal{Z}}(P^m) \neq \emptyset\}| \leq K$.*

PROOF SKETCH. The disjoint covering condition ensures that each region is still part of exactly one partition. The union condition guarantees that the new set of partitions don't "cover" different sets of region; rather, there is exactly one merged partition that is responsible for all regions from an original partition. Finally, the size condition imposes the constraint that each merged partition overlaps with at most K regions from the geoset; therefore, a solution based on the merged partitions can be mapped equivalently to a solution on the original set of partitions. □

The intuition underlying Lemma 4.2 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 during Algorithm CPALGO, i.e., in a single pass of the data: (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 4.3. *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 4.2 and runs in one pass of the spatial tree.*

PROOF SKETCH. It is easy to see that Algorithm 1 traverses the spatial tree only once, since it performs a depth-first traversal of the tree. Further, the algorithm constructs a merged partition only if the size condition is satisfied; and since the merged partition is always constructed as a union of existing partitions, the Union condition of Lemma 4.2 is satisfied. Finally, if a merged partition is constructed, Algorithm 1 doesn't traverse child nodes, thereby ensuring disjointness. □

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 4.4 (*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 4.2), 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 $c_{q_2} \in span_{\mathcal{Z}}(P_{q_2})$, $q_1 \neq q_2$, such that c is the least-common ancestor of c_{q_1}, c_{q_2} in T.

Intuitively, a node c is a critical node if it is the 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 4.5 (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.).*

PROOF SKETCH. The result follows from the fact that a constraint on any particular node $c \in \mathcal{N}$ in the spatial tree is identical to the constraint on the critical node $c' \in \mathcal{N}$ in c's subtree with maximum height. That is, c' is the critical node below c that is closest to c. Note that there is a unique closest critical node c' below c: if not, c would have been a least-common ancestor and be a critical node itself. □

Bounded Cover of Regions. While Definition 3.3 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 [Gonzalez et al. 2010]. The bounded cover of size M for every region imposes a bound on the number of critical nodes.

LEMMA 4.6. *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 $nM\mathcal{Z}$.*

PROOF SKETCH. Every critical node of the tree must be an ancestor of some node in the bounded cover of at least one region. Since there are at most nM bounded covers, there are at most $nM\mathcal{Z}$ nodes that are candidates for being critical nodes. □

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

THEOREM 4.7. *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 $|\mathcal{C}| = \mathcal{O}(nM\mathcal{Z})$.*

PROOF SKETCH. Follows from Theorem 4.1 and Lemmas 4.2, 4.3, 4.5, and 4.6. □

4.1.3. 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 $V(P_q) = \sum_z \sum_{z^* \leq z} v_q^{z^*}$, i.e., we compute the number of records from P_q visible at some cell c_j^z, and aggregate 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 4.8. Continuing with the solutions in Example 3.8 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 II 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 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.

Table II. Table comparing the objective measures for various solutions in Example 4.8.

	$\mathcal{F}_{max}$	$\mathcal{F}_{distinct}$	$\mathcal{F}_{fair}$	$\mathcal{F}_{imp}$
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

4.2. 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 9, we show that in practice $\mathbb{P}^r$ gives the optimal solution in all real datasets.

5. 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 [Gonzalez et al. 2010].[3]

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

5.1. Strong and weak maximality

Maximality can be defined as follows.

Definition 5.1 (*Strong Maximality*). A solution $M : \{1, \ldots, n\} \rightarrow \{1, \ldots, \mathcal{Z}, \mathcal{Z}+1\}$ 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 3.2.2 ensures strong maximality (but strong maximality doesn't ensure optimality in terms of $\mathcal{F}_{max}$).

Example 5.2. Recall the data from Figure 3, and consider solutions M^1, M^2, M^3 and M^4 from Example 3.8 and 4.8. 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 3.8 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.

THEOREM 5.3 (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.*

PROOF SKETCH. We give a reduction from the NP-hard EXACT SET COVER problem [Garey and Johnson 1979]: Given a universe $U = \{1, \ldots, n\}$ of n elements and a family $\mathcal{S} = \{S_1, \ldots, S_m\}$ of subsets of U, determine if there exists a subset $\mathcal{S}^* \subseteq \mathcal{S}$ such that: (1) $U = \bigcup_{S \in \mathcal{S}^*} S$, (2) $\forall S_i, S_j \in \mathcal{S}^*$, $S_i \neq S_j \Rightarrow (S_i \cap S_j) = \emptyset$.

Given an instance of the Exact Set Cover problem, we construct an instance of the thinning problem as follows: Construct a spatial tree with $\mathcal{Z} = \lceil \log_4 n \rceil$ levels, n special leaf nodes $c_1^{\mathcal{Z}}, \ldots, c_n^{\mathcal{Z}}$. Also, we construct a geoset with m records $G = \{R_1, \ldots, R_m\}$, where the region of R_i is defined by exactly the set of leaf nodes corresponding to elements covered by S_i: R_i spans cell $c_j^{\mathcal{Z}}$ if and only if $j \in S_i$. Finally, we set $K = 1$.

We claim that the strongly maximal solution to this instance of the thinning problem has exactly one record visible at each of the n cells $c_1^{\mathcal{Z}}, \ldots, c_n^{\mathcal{Z}}$: (1) Let S^* be a solution to the exact set cover problem; then we can set $M(i) \leq \mathcal{Z}$ if and only if $R_i \in S^*$, and set $M(i) = (\mathcal{Z} + 1)$ otherwise. (The exact assignment of a value between 0 and $\mathcal{Z}$ for each $M(i)$ with $R_i \in S^*$ is irrelevant as we can just pick any arbitrary assignment ensuring all ancestors of each $c_j^{\mathcal{Z}}$, $j \leq n$, have exactly one visible record. Note that this is a strongly maximal solution to thinning since all possible leafs (and all their ancestors) have exactly one record. (2) Conversely, if there is a solution M that ensures every leaf node $c_j^{\mathcal{Z}}$, $j \leq n$ has one visible record, then the sets corresponding to all these visible records constitute a solution to the exact set cover problem. □

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 5.4 (*Weak Maximality*). A solution $M : \{1, \ldots, n\} \rightarrow \{1, \ldots, \mathcal{Z}, \mathcal{Z}+1\}$ 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 $M' : \{1, \ldots, n\} \rightarrow \{1, \ldots, \mathcal{Z}, \mathcal{Z}+1\}$ 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 5.5. Continuing with Example 5.2, we can see that M^2 (defined in Example 3.8) 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 3.8 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 3.2.2.

LEMMA 5.6. *Consider a thinning solution $M : \{1, \ldots, n\} \rightarrow \{1, \ldots, \mathcal{Z}, \mathcal{Z}+1\}$ 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.*

PROOF SKETCH. We prove each part of the result separately:

— Suppose the solution based on $\mathcal{F}_{max}$ is not strongly-maximal. Then based on the violation of Definition 5.1, we can find a solution M' which has as many records visible at every cell, and more records visible at at least one cell. Based on Theorem 4.1, M' satisfies all constraints of the integer program. Since $\mathcal{F}_{max}$ aggregates the counts of each cells, M' gives a higher objective value than M, leading to a contradiction.
— Follows directly from Definitions 5.1 and 5.4.
— Suppose a dataset consisted only of points, and we have a weakly-maximal solution M. Suppose that M is not strongly-maximal. Then based on Definition 5.1, there exists some node $c \in \mathcal{N}$ in the strongly-maximal solution, for which fewer records are visible in the weakly-maximal solution. Let us consider such a cell c in more detail. Consider the set of records $Vis_M(G, T, c)$ visible at c based on the weakly-maximal solution M. To be able to increase the visibility count of this cell in the strongly-maximal solution, there must be at least one region $R \notin Vis_M(G, T, c)$ that spans c. Since this region R is currently not visible in cell c, and the current visibility count at c is less than K (since the strongly-maximal solution increases its count), we can safely set $M(i)$ to z, where z is the zoom level of c. This revision to M is a violation of Definition 5.4.

□

5.2. 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 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 $M : \{1, \ldots, n\} \rightarrow \{1, \ldots, \mathcal{Z}+1\}$.
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.

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

(1) **Correctness:**
  — **Weak Maximality:** The weak-maximality of Algorithm 2 follows from the DFS tree traversal: Every single cell c of the tree is considered before all of its de-

scendants. And when cell c is considered, as many records as possible are made visible at the given cell. Therefore, no record at a descendant cell c' can be made visible at c (otherwise it would have been added when c was considered), giving us the necessary condition of Definition 5.4.

— **Strong Maximality for Points:** Follows from weak maximality and Lemma 5.6.

(2) **Complexity:** Note that in a DFS traversal of a 4-ary tree with height $\mathcal{Z}$, the stack size never grows more than $4\mathcal{Z}$, giving the space requirement of $4\mathcal{Z}$. Also, note that each record is considered once for every zoom level between 0 and $\mathcal{Z}$, giving a time complexity of $n\mathcal{Z}$.

□

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

Example 5.8. 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 5.5).

6. 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 3) 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 $\mathcal{I} : \mathcal{N} \rightarrow 2^G$, 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 [Hilbert 1891; Guttman 1984]) in $O(n \log n)$ with one scan of the entire dataset. Assignment of the zoom level then requires one index scan.

THEOREM 6.1 (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 3 returns a strongly maximal solution to the thinning problem with an offline computation time $O(n(\mathcal{Z} + \log n))$, and constant (independent of the number of points) memory requirement.*

PROOF SKETCH. It is easy to see that the solution returns a strongly maximal solution *if* it is a thinning solution: For each record we show up to K points if possible, so there is no room to increase the visibility count for any cell. The more subtle aspect of the result is the fact that the algorithm indeed returns a thinning solution, in particular, that it satisfies the zoom consistency. To ensure zoom consistency, we note that the random number assignment gives a global priority ordering of all regions; hence, if a point has higher priority at some cell c, then it also have a higher priority at de-

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 $M : \{1, \ldots, n\} \rightarrow \{1, \ldots, \mathcal{Z}+1\}$.
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.

scendant/ancestor cells, ensuring zoom consistency. Finally, the offline computation is performed once for each point for each zoom level. □

Furthermore, Algorithm 3 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. One way of implementing the retrieval of first K points for a given cell is to apply a post-filtering step after the index retrieval. In this case, the first step of random number assignment can be performed online as well, completely eliminating offline processing.
3. 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).
4. 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.

7. CONTIGUOUS REGIONS

So far we have considered arbitrary regions (Definition 3.3) that may span any subset of leaf cells in the spatial tree. In particular, regions may consist of cells from completely different parts of the world. However, a common special case is that of "contiguous regions", which represent regions that are not divided in space. In this section, we discuss how our results apply to the special case of contiguous regions.

A contiguous region over spatial tree $T(\mathcal{Z}, \mathcal{N})$ with nodes at level-$\mathcal{Z}$ being $N^{\mathcal{Z}} = \{c_1^{\mathcal{Z}}, \ldots, c_{4^{\mathcal{Z}}-1}^{\mathcal{Z}}\}$ may have a contiguous region that does not span consecutive cells from $N^{\mathcal{Z}}$, as shown by the example below.

Example 7.1. Consider region $R4$ from Figure 3(b), which is contiguous in space. However, $R4$ would be represented using cells $S4 = \{c_1^3, c_3^3\}$ based on Definition 3.3, which do not constitute a consecutive set of cells. Therefore, $R4$ is an example of a region that is not contiguous based on Definition 3.3.

A complete characterization of the set of leaf cells that represent a contiguous region is outside the scope of this paper. Instead, we use a simplified definition for contiguous regions:

Definition 7.2 (*Contiguous Region*). Consider a region $R(S)$ over a spatial tree $T(\mathcal{Z},\mathcal{N})$ defined by a subset $S \subseteq \mathcal{N}^{\mathcal{Z}}$, and let $c \in \mathcal{N}$ be the least common ancestor of nodes in S. We say that $R(S)$ is a *contiguous region* if and only if $\forall c^{\mathcal{Z}}$ descendant of c, we have that $c^{\mathcal{Z}} \in S$.

We use the shorthand $R^*(c)$ to denote the contiguous region $R(S)$, where S is the set of all of c's descendant leaf nodes in $T(\mathcal{Z},\mathcal{N})$.

Intuitively, a contiguous region $R(S)$ (or $R^*(c)$) must be represented by a subset of cells that completely cover the leaf nodes of a subtree rooted at some internal node c. Next we investigate how contiguous regions affect the results obtained in the rest of the paper.

We start by investigating the implications of contiguous regions on maximality (Section 7.1), and then briefly discuss the integer programming formulation (Section 7.2).

7.1. Maximality

Our main result of this section shows that when all regions are contiguous, strong maximality is in PTIME. Contrast this with the NP-hardness for the general case (Theorem 5.3).

THEOREM 7.3 (TRACTABILITY OF STRONG MAXIMALITY). *Given a geoset $G = \{R_1, \ldots, R_n\}$ consisting of contiguous regions over a spatial tree $T(\mathcal{Z},\mathcal{N})$, and a maximum bound $K \in \mathbb{N}$, the problem of finding a strongly maximal solution to the thinning problem is in PTIME.*

In the following, we shall develop an algorithm that achieves strong maximality in polynomial time, thereby proving the result. Let us start with a definition of "domination" between a pair of contiguous regions. Recall we use the shorthand $R^*(c)$ to denote a contiguous region defined by all leaf node descendants of an internal node c.

Definition 7.4 (*Domination*). Given contiguous regions $R_1^*(c_1)$ and $R_2^*(c_2)$ over spatial tree $T(\mathcal{Z},\mathcal{N})$, we say that $R_1^*(c_1)$ *dominates* $R_2^*(c_2)$ if c_1 is an ancestor of c_2 in T.

We have the following straightforward observation about domination of contiguous regions:

LEMMA 7.5 (DOMINATION). *Given contiguous regions $R_1^*(c_1)$ and $R_2^*(c_2)$ over spatial tree $T(\mathcal{Z},\mathcal{N})$, if $c_1 \neq c_2$, exactly one of the following properties holds:*

(1) $R_1^*(c_1)$ *dominates* $R_2^*(c_2)$
(2) $R_2^*(c_2)$ *dominates* $R_1^*(c_1)$
(3) $(span(R_1^*(c_1) \cap span(R_2^*(c_2))) = \emptyset$

PROOF SKETCH. Given two distinct nodes c_1 and c_2 in the spatial tree T, either c_1 is an ancestor of c_2, or c_2 is an ancestor of c_1, or the subtrees rooted at c_1 and c_2 have a disjoint set of nodes. □

The lemma above is based on the observation that if $c_1 \neq c_2$, either c_1 is an ancestor of c_2, or c_2 is an ancestor of c_1, or they don't share any descendant.

Next we present a set of results on any thinning solution that enable us to obtain a polynomial-time algorithm:

LEMMA 7.6 (WEAK MAXIMALITY). *Given a geoset $G = \{R_1, \ldots, R_n\}$ consisting of contiguous regions over a spatial tree $T(\mathcal{Z},\mathcal{N})$, a maximum bound $K \in \mathbb{N}$, and a thin-*

ning solution $M : \{1, \ldots, n\} \rightarrow \{1, \ldots, \mathcal{Z}, \mathcal{Z}+1\}$. Let the internal node defining R_i be $c_i^{z_i}$, i.e., R_i is the contiguous region defined by an internal node c_i at zoom level z_i. If $z_i < M(i) < (\mathcal{Z}+1)$, then M is not weakly maximal.

PROOF SKETCH. Consider M' obtained from M as follows: (a) $\forall j \neq i : M'(j) = M(j)$, (b) $M'(i) = z_i$. We claim that M' is a thinning solution, thus violating the weak maximality condition in Definition 5.4. A key observation to showing that M' is a thinning solution is that when all regions are contiguous, we have that the number of regions visible at any cell c is at most as many as those visible at some descendant c' of c: This property holds because all regions that span c also span its descendant c'. Therefore, suppose M' violates the visibility bound of some cell $c*^{z'}$, for $z' \geq z_i$. We can now find a cell in M that also violates the visibility bound. Let the original value of $M(i)$ be x. Now, consider the descendant $c*^{x}$ of $c*^{z'}$ at zoom level x: M must violate the visibility bound of $c*^{x}$ since all regions that are visible at $c*^{z'}$ based on M' are also visible at $c*^{x}$ based on M, contradicting the fact that M is a thinning solution. □

Intuitively, the result above says that when all regions are contiguous, any weak maximal solution sets $M(i)$ for a region R_i to be either $\mathcal{Z}+1$ (i.e., not visible at all), or sets $M(i)$ to at most z_i (i.e., visible at all cells spanned by R_i). Therefore, finding a weakly maximal thinning solution reduces to the problem of: (1) finding a maximal subset $G_s \subseteq G = \{R_1, \ldots, R_n\}$ of regions, all of which are visible at each of the spanned cells, (2) restricting to records in G_s and finding any weakly maximal solution among them. The second step above can reuse the techniques from Section 5. Henceforth, we focus on Step (1) and simply use the shorthand $M(G_s)$ to represent the thinning solution with: (a) $\forall R_i \in G_s : M(i) \leq z_i$ as determined by Step (2), (b) $M(i) = (\mathcal{Z}+1)$ otherwise.

We are now ready to present our main test for strong maximality.

THEOREM 7.7 (STRONG MAXIMALITY CONDITION). *Given a geoset $G = \{R_1, \ldots, R_n\}$ consisting of contiguous regions over a spatial tree $T(\mathcal{Z}, \mathcal{N})$, a maximum bound $K \in \mathbb{N}$, a weakly maximal thinning solution $M(G_s \subseteq G)$ is strongly maximal if and only if the following condition holds: for any two distinct regions $R_i \in G_s$ and $R_j \in (G \setminus G_s)$, R_j does not dominate R_i.*

PROOF SKETCH. We prove the necessity and sufficiency in two parts:

— **Only-if:** Suppose that for some $R_i \in G_s$ and $R_j \in (G \setminus G_s)$, we have that R_j dominates R_i. Let Dom be the set of all regions in G_s such that for each $R \in Dom$: (1) R_j dominates R, (2) For any $R' \neq R$, $R' \in G_s$, R' does not dominate R. Intuitively, Dom is the set of all regions in G_s dominated by R_j such that there are no domination relationships among them; if there are two regions R and R' dominated by R_j, if R dominates R', then only R is added to Dom. We have that the thinning solution $M'(G_s \setminus Dom \cup \{R_j\})$ violates the strong maximality of M. First, note that M' is indeed a thinning solution: the set of regions in Dom collectively contribute a visibility of at most 1 to each cell spanned by R_j. Further, R_j adds a visibility of 1 to some extra nodes (e.g., those on the path from R_i to R_j), without violating the visibility bound (as shown in Lemma 7.6).

— **If:** Next we show that if for any two distinct regions $R_i \in G_s$ and $R_j \in (G \setminus G_s)$, R_j does not dominate R_i, then $M(G_s)$ is strongly maximal if it is weakly maximal. Suppose $M(G_s)$ is not maximal, consider a different strong maximal solution $M'(G'_s)$. By Definition 5.1, we must have some cell c for which M' gives a higher visibility count than M. Consider the cell c in the thinning solution $M(G_s)$. If we can obtain a higher visibility on cell c, then there must be a region $R^*(c*) \in (G \setminus G_s)$ that spans c. Since $M(G_s)$ is weakly maximal, adding R^* to G_s violates the visibility of some cell c' that is a descendant of c, since otherwise the same visibility bound of c would

Algorithm 4 An algorithm that returns a strongly-maximal thinning solution for geosets of contiguous regions.

1: **Input:** Geoset $G = \{R_1 * (c_1^{z_1}), \ldots, R_n * (c_n^{z_n})\}$ of contiguous regions over spatial tree $T(\mathcal{Z}, \mathcal{N})$, spatial index $\mathcal{I}$ visibility bound $K \in \mathbb{N}$.
2: **Output:** Strongly-maximal Thinning Solution $M(G_s)$
3: Initialize $G_s \leftarrow \emptyset$.
4: Let ND be the non-dominated regions in $(G \setminus G_s)$
5: **for** $R \in ND$ **do**
6: **if** (adding R to G_s does not violate visibility) **then**
7: $G_s = G_s \cup \{R\}$
8: Set ND to non-dominated regions in $(G \setminus G_s)$
9: Continue;
10: **Return** M.

have been violated. Since c''s visibility bound is violated by including $R^*(c*)$ based on some region $R' \in G_s$ that is dominated by $R^*(c*)$, the pair R', R^* violates the sufficiency condition of our theorem.

□

Based on Theorem 7.7, our goal is reduced to finding a maximal subset $G_s \subseteq G = \{R_1, \ldots, R_n\}$ such that: (1) the visibility bound of all cells are satisfied; (2) no region $R_i \in G_s$ is dominated by a region $R_j \in (G \setminus G_s)$. We can obtain such a G_s by considering regions in order of their domination strength: we start with regions that are not dominated by any other region and start including them into G_s as long as the visibility bound of cells is not violated. As regions are added into G_s, we keep considering regions that are non-dominated in $(G \setminus G_s)$. Algorithm 4 gives the basic pseudo-code for this algorithm, where the computation of the non-dominated set of regions ND is described in one step. In order to efficiently pick non-dominated regions from $(G \setminus G_s)$, we may create an efficient forest representation of the partial-order imposed by the domination relationship; for instance, we may have a collection of binary search trees, with the comparator being the domination relationship. Details of this implementation are omitted.

We conclude by presenting the main result of this section.

THEOREM 7.8 (STRONG MAXIMALITY ALGORITHM). *Given a geoset $G = \{R_1, \ldots, R_n\}$ consisting of contiguous regions over a spatial tree $T(\mathcal{Z}, \mathcal{N})$, and a maximum bound $K \in \mathbb{N}$, Algorithm 4 returns a strongly-maximal solution $M(G_s)$ to the thinning problem in $O(n(n + \mathcal{Z}))$.*

PROOF SKETCHES OF THEOREMS 7.3 AND 7.8. Follows from Algorithm 4 and Theorem 7.7: Note that Algorithm 4 adds non-dominated regions whenever visibility is not violated, ensuring weak maximality. Further, since any region R is added before every region R' that R dominates, the condition of Theorem 7.7 is satisfied. □

7.2. Integer programming

Since we have shown that strong maximality is achievable in polynomial-time, we consider the integer programming only very briefly, describing how our results from Section 4 change. Recall the definition of bounded covers from Section 4.1.2. By definition, a contiguous region $R * (c)$ has a bounded cover of size 1 (the cover c). Therefore, we obtain the following corollary of Theorem 4.7:

COROLLARY 7.9. *Given a geoset $G = \{R_1, \ldots, R_n\}$ of contiguous regions 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* $|\mathcal{C}| = \mathcal{O}(n\mathcal{Z})$.

PROOF SKETCH. Directly follows from Theorem 4.7 based on the fact that a contiguous region has a cover of size 1. □

8. INTEGER PROGRAMMING FOR POINTS

Recall, in Section 4.2 we considered a relaxed program $\mathbb{P}^r$ that is obtained by eliminating the integer constraints (Equation (3)) on v_q^z's. The relaxation enabled us to solve the problem efficiently sacrificing optimality, since integer programs are known to be much harder to solve than linear programs. This section closely explores the structure of our program to understand what makes the problem hard. In particular, the main result we develop in this section is to show that when our dataset consists only of points, the relaxed linear program is in fact optimal.

To study properties of our linear relaxation of the integer program, we consider an equivalent form of the relaxed program $\mathbb{P}^r$. Note that $\mathbb{P}^r$ is the same as $\mathbb{P}$ in Section 4.1 except $\mathcal{C}^r$, which does not have the integer constraints (Equation (3)). With the linear objective functions in Section 4.1.3, we first define a linear program $\mathbb{P}'(\mathcal{P}, \mathcal{W}, \mathcal{D}^r)$ from $\mathbb{P}^r(\mathcal{P}, \mathcal{V}, \mathcal{C}^r)$ as follows.

Variables of the transformed linear program: The set of variables $\mathcal{W}$ in the program $\mathbb{P}'$ is obtained from the same partitions $\mathcal{P}$: For each partition P_q, we construct $\mathcal{Z}$ variables $w_q^1, w_q^2, \ldots, w_q^{\mathcal{Z}}$, such that $w_q^z \equiv \sum_{z^*=1}^{z} v_q^{z^*}$. Intuitively, w_q^z represents the *number* of records from partition P_q whose min-levels are smaller than or equal to z, that is, w_q^z is the number of records from P_q that are visible at level z.

Constraints: The set $\mathcal{D}^r$ of constraints are:

$$w_q^{\mathcal{Z}} \leq |P_q| \tag{7}$$

$$\forall q \forall z < \mathcal{Z} : w_q^z \leq w_q^{z+1} \tag{8}$$

$$\forall q \forall z : w_q^z \geq 0 \tag{9}$$

$$\forall c_j^z \in \mathcal{N} : \sum_{q : c_j^z \in span(P_q)} w_q^z \leq K \tag{10}$$

To study this linear program, we define a specific encoding of the above linear program using the following matrix notation.

Definition 8.1 (*Matrix Representation of* $\mathbb{P}'(\mathcal{P}, \mathcal{W}, \mathcal{D}^r)$). We define $\mathbb{P}'(\mathcal{P}, \mathcal{W}, \mathcal{D}^r)$ in matrix notation as follows.

$$\begin{aligned} \text{maximize } \mathcal{F} &= c^T \mathbf{w} \\ \text{subject to } A\mathbf{w} &\leq \mathbf{b} \\ \text{and } \mathbf{w} &\geq 0 \end{aligned}$$

A is the constraint matrix enumerating: (1) constraints on the variable size (Constraints (7) $\sim$ (9)) in ascending order of q and z, and then (2) constraints on the cells (Constraint (10)) in ascending order of z. With this construction, $\mathbf{w}$ and $\mathbf{b}$ are as follows:

$$\mathbf{w} = (w_1^1, w_1^2, \cdots, w_1^{\mathcal{Z}}, w_2^1, \cdots, w_2^{\mathcal{Z}}, \cdots, w_{|\mathcal{P}|}^1, \cdots, w_{|\mathcal{P}|}^{\mathcal{Z}})^T,$$

$$\mathbf{b} = (0, 0, \cdots, |P_1|, 0, \cdots, |P_2|, \cdots, 0, \cdots, |P_{|\mathcal{P}|}|, K, \cdots, K)^T.$$

c^T is a vector of coefficients that defines $\mathcal{F}$.

Let $\mathbf{w}^*$ be a solution of $\mathbb{P}'$, and $\mathbf{v}^*$ be the corresponding solution of $\mathbb{P}^r$. The following equation gives $\mathbf{v}^*$ from $\mathbf{w}^*$:

$$\mathbf{v}^* = L\mathbf{w}^*$$

where L is a lower triangular matrix whose main diagonal entries are $+1$, entries right below them are -1. All other entries are 0.

To understand the structure of A, we define partitions of it, based on the constraints they represent:

Definition 8.2 (*Partitioning of* A). We partition A vertically into two submatrices B and C. B is the upper block of A corresponding to constraints on variable size (Constraints (7) $\sim$ (9)). C is the lower block of A corresponding to constraints on cells (Constraints (10)).

$$A = \left(\frac{B}{C}\right)$$

Example 8.3. Figure 4 presents a structural overview of A. Submatrices B and C are distinguished by double horizontal lines. Single lines are to help identify blocks by q and z. '?' represents that the value can be either 0 or 1 depending on the specific instance of the problem.

We make the following observations on the structure of A, which follow from the construction of A in Definition 8.1:

(1) B is a square matrix of order $\mathcal{Z}|\mathcal{P}|$.
(2) B is an upper triangular matrix whose main diagonal entries are $+1$ and entries right above them, except those at column $j\mathcal{Z}+1, 1 \leq j < |\mathcal{P}|$, are -1. All other entries are 0.
(3) Each column of C has at most one $+1$, and all other entries are 0. This follows from the fact that z is fixed given a column and a point record can span only a single cell at level z.

We now state a classical result that connects the matrix representation to linear programming: A matrix M is *totally unimodular* if the determinant of every square submatrix of M is 0, -1 or $+1$. Its important connection to linear programming is that if the constraint matrix is totally unimodular and bounds are integral, solving a continuous relaxation of the problem always yields an integral solution vector, if any exists [Schrijver 1986].

We are now ready to state the main result of this section in the following theorem, whose proof involves showing that our constraint matrix A is totally unimodular based on the properties (1) $\sim$ (3).

THEOREM 8.4 (OPTIMALITY OF LINEAR RELAXATION). *If a geoset* $G = \{R_1, \ldots, R_n\}$ *consists of only point records, the relaxed linear program* $\mathbb{P}^r$ *defined in Section 4.2 is integral, and thus,*

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

where $\mathbb{P}$ *is the integer programming formulation.*

PROOF SKETCH. Consider an arbitrary square submatrix A' of A. The goal is to prove $det(A') \in \{\pm 1, 0\}$, and thus A' is totally unimodular. We prove this by induction on the order t of $\mathbf{A}'$. The base case of $t = 1$ is trivial so we assume $t > 1$.

Case 1: $\mathbf{A}'$ has a column with only zeros, then $det(A') = 0$.

$$A = \left(\begin{array}{cccc|cccc|ccc}
1 & -1 & \cdots & 0 & 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\
0 & 1 & -1 \cdots & 0 & 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 1 & 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\
\hline
0 & 0 & \cdots & 0 & 1 & -1 \cdots & 0 & \cdots & 0 & \cdots & 0 \\
0 & 0 & \cdots & 0 & 0 & 1 \cdots & 0 & \cdots & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 0 & 0 & \cdots & 1 & \cdots & 0 & \cdots & 0 \\
\hline
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\
\hline
0 & 0 & \cdots & 0 & 0 & 0 \cdots & 0 & \cdots & 1 & -1 \cdots & 0 \\
0 & 0 & \cdots & 0 & 0 & 0 \cdots & 0 & \cdots & 0 & 1 \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 0 & 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \\
\hline
? & 0 & \cdots & 0 & ? & \cdots & 0 & \cdots & ? & \cdots & 0 \\
\hline
0 & ? & \cdots & 0 & 0 & ? \cdots & 0 & \cdots & 0 & ? \cdots & 0 \\
0 & ? & \cdots & 0 & 0 & ? \cdots & 0 & \cdots & 0 & ? \cdots & 0 \\
0 & ? & \cdots & 0 & 0 & ? \cdots & 0 & \cdots & 0 & ? \cdots & 0 \\
0 & ? & \cdots & 0 & 0 & ? \cdots & 0 & \cdots & 0 & ? \cdots & 0 \\
\hline
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\
\hline
0 & 0 & \cdots & ? & 0 & 0 \cdots & ? & \cdots & 0 & 0 \cdots & ? \\
0 & 0 & \cdots & ? & 0 & 0 \cdots & ? & \cdots & 0 & 0 \cdots & ? \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & ? & 0 & 0 \cdots & ? & \cdots & 0 & 0 \cdots & ?
\end{array}\right)$$

Fig. 4. Structure of the constraint matrix A from Definitions 8.1 and 8.2

Case 2: $\mathbf{A}'$ has a column with exactly one non-zero. After permuting rows and columns we have

$$A' = \begin{pmatrix} \pm 1 & a^T \\ 0 & A'' \end{pmatrix}$$

for some vector a and matrix A''. By the induction hypothesis, $det(A'') \in \{\pm 1, 0\}$, hence $det(A') \in \{\pm 1, 0\}$.

Case 3: Each column of A' has two or more non-zeros. We show that this case does not exist from our construction of A.

Suppose that A has a square submatrix A' such that all columns have at least two non-zeros. First, we claim that A' is not a submatrix of either B or C. Suppose A' is a submatrix of B and its order is t. We note that A' cannot have any empty row (rows with only zeros) since there are at least $2t$ non-zeros in A' from the assumption, which requires at least t non-empty rows. This is because rows of B have at most two non-zeros ($2t/2 = t$). Thus, $A'_{0,0}$ must be -1 for its first column to have two non-zeros, and the first row has a single non-zero. We observe that the last row of A' also has a single non-zero, i.e. $+1$, otherwise the last column cannot have at least two non-zeros. Since there are at least two rows with a single non-zero, the number of rows of A' must be at least $t + 1 = (2t - 2)/2 + 2$. This contradicts that A' is a square matrix, and thus A' cannot belong to B. Each column of C has at most a single $+1$, thus C cannot contain A' which has at least two non-zeros at all of its columns.

Second, suppose A' spans both B and C. We show that A' has at least t rows in B, which contradicts that A' is a square matrix. Pick $g, g \leq t$, columns in any single vertical block that represents variables of the same record (c.f. Figure 4). If those g columns don't have any non-zero in C, with the similar reasoning as above, they occupy at least $g + 1$ rows in B. If there is a column which spans C, they occupy at least g rows in B since the column needs at least one non-zero in B and the rows that span them can overlap with other rows with non-zeros. Since rows with non-zeros in B from different vertical blocks are distinct, A' has at least t rows in B and A' cannot be a square matrix.

Since any square submatrix in A has a determinant of 0 or ± 1, A is totally unimodular and this concludes the proof. □

The implication of Theorem 8.4 is that we can obtain an efficient solution for point datasets without hurting optimality: we just need to solve the linear program, which is typically much more efficiently solvable than integer programs. We shall empirically illustrate the general equivalence of the two programs for specific datasets in Section 9.3.

9. EXPERIMENTS

This section presents a detailed experimental evaluation of our algorithms. After presenting our datasets and experimental setup in Section 9.1, we present the following main experimental findings:

1. In Section 9.2, we show that the optimization program minimization techniques from Section 4.1.2 usually reduce the size of the problem by more than two orders of magnitude.
2. In Section 9.3, we show that in all seven of our datasets, the integer relaxation (Section 4.2) doesn't affect optimality as compared to the integer formulation.
3. Section 9.4 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 9.5, 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 9.6 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.

9.1. 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 [Gonzalez et al. 2010].

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 [Olson et al. 2001], (6) points from 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 4.1.2. Each record forms a single partition.
- Opt_{max} is the algorithm described in Section 4 with objective $\mathcal{F}_{max}$ in Equation (5).
- Opt_{imp} is the algorithm described in Section 4 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 3, which works for point datasets.

We use Opt_{naive} only to demonstrate how well the optimization framework can scale without the minimization technique. We implemented $Rand$ with a post-filtering step that selects first K points at a cell as described in Section 6. That is, given a cell to render, we retrieve points that belong to the cell from $\mathcal{I}$, assign numbers uniformly at random (but deterministically to maintain consistency across independent views), and select the first K points. Since $Rand$ only needs to assign random numbers to records and does not involve any 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[4] for our linear optimization. The solver is a linear programming (LP) solver. We relaxed the integer constraints as proposed in Section 4.2 and rounded down solutions from the solver; an optimal integer solution would require using a powerful integer solver such as CPLEX [5]. 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 experiments 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.

9.2. Benefit of minimizing program size

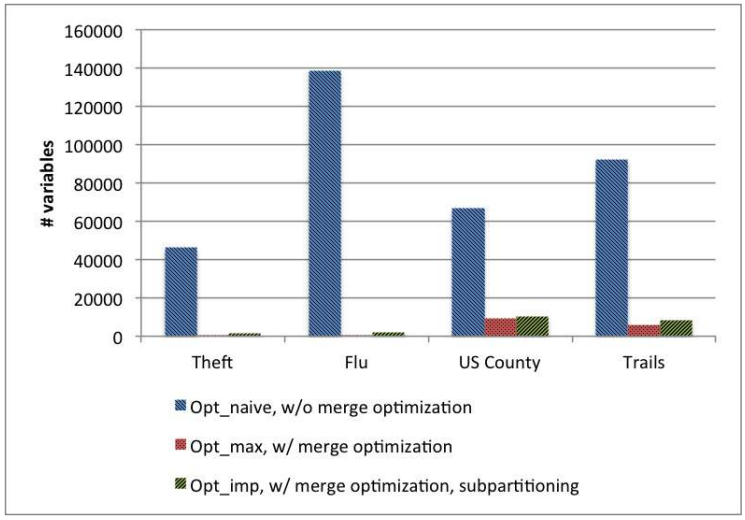
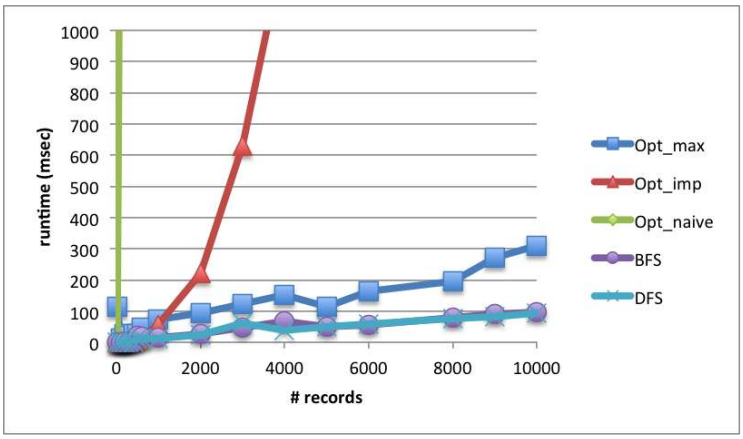


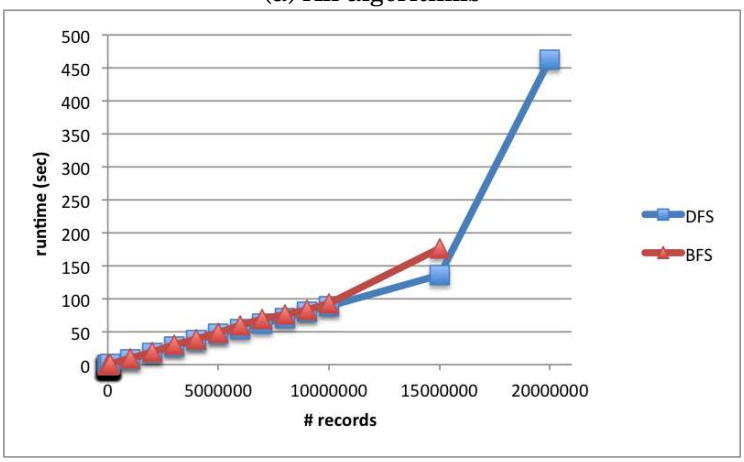
Fig. 5. Impact of Merging Partitions

We show effectiveness of the program size minimization techniques in Section 4.1.2. Figure 5 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 4.1.2. 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.

[4]http://commons.apache.org/math/
[5]http://www-01.ibm.com/software/integration/ optimization/cplex-optimizer/



(a) All algorithms



(b) BFS & DFS

Fig. 6. Scalability

9.3. Integer relaxation

We compared our integer program solution with the relaxed solution (Section 4.2). 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, thereby behaving almost like point datasets (which we know is integral from Section 8). This shows that employing the relaxed solution does not affect optimality (significantly).

9.4. Scalability

We study scalability using the US Parcel dataset, which is our largest dataset. Figure 6 plots runtime versus the number of records. To properly show the scale, Figure 6(a) plots a small data size range (up to 100,000 records), and Figure 6(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. 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 while BFS fails to scale up to that many 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.

9.5. Objectives

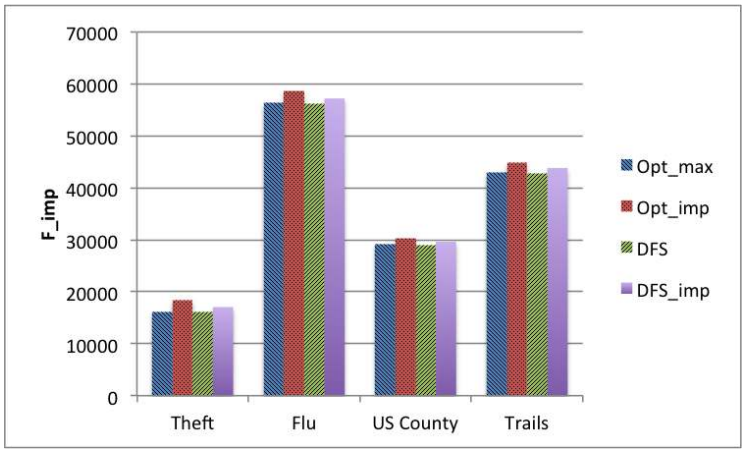


Fig. 7. Objective Based on Importance with Uniform Distribution

First we consider optimality in datasets with importance. Figure 7 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 in Figure 8, 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.

Our observation on $\mathcal{F}_{max}$ is also very similar. $\mathcal{F}_{max}$ of DFS is within 1% of Opt_{max} for all datasets. 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 9(a) shows the result from maximizing $\mathcal{F}_{max}$ at a particular zoom level where all of North America is on the map. The figure shows a thinning solution by representing each visible record using a red dot. (The overlap of red dots is simply because of the rendering on the map.)

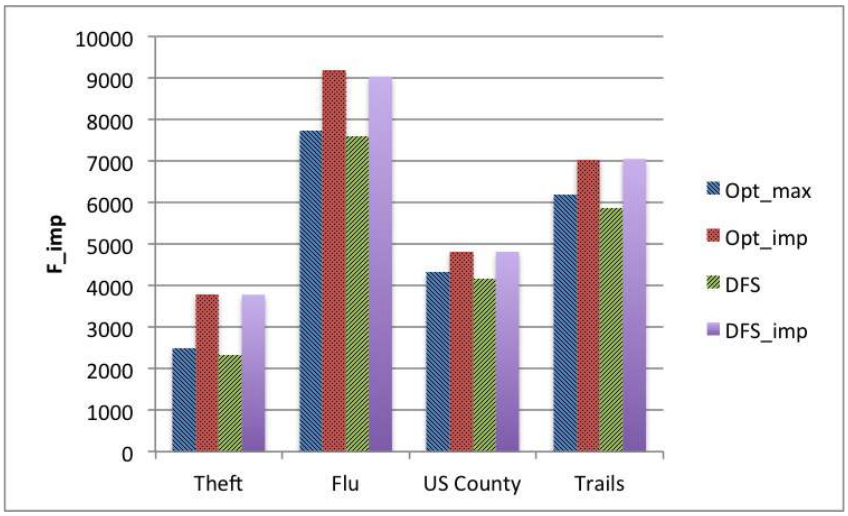
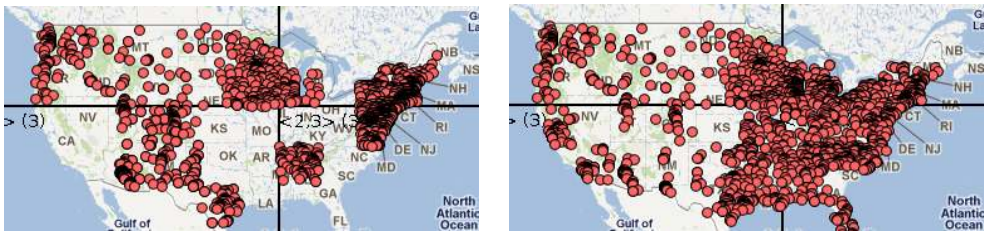


Fig. 8. Objective Based on Importance with Zipfian Distribution



(a) Flu with F_{max} (b) Flu with F_{imp}

Fig. 9. Results with Different Objective Functions

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 9(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 [6].

9.6. Optimization runtime

Figure 10 presents the break-down of runtime of each of the optimization programs. The build time includes creation of Cover and Touch data structures in Algorithm 1, and the solve time is the time taken by the solver to solve the LP. The sample time is the time taken for actually applying the solution to

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

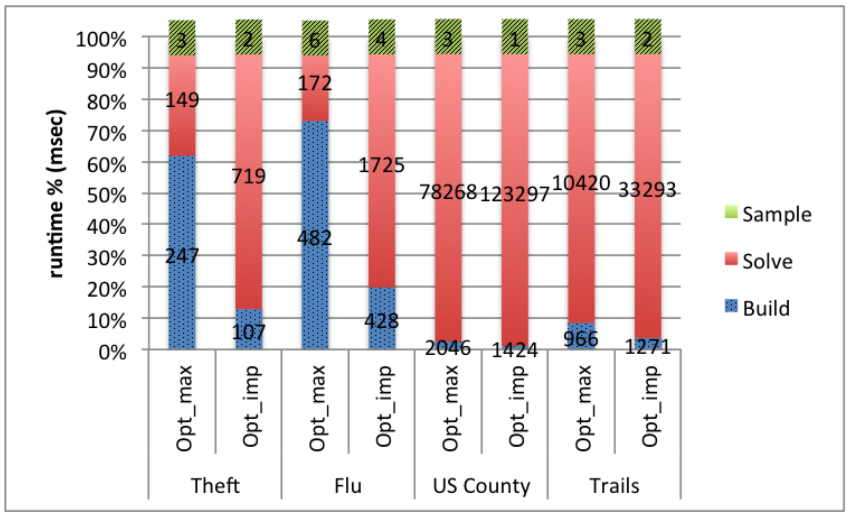


Fig. 10. Breakup of Runtime

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.

10. 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 [Gonzalez et al. 2010], demonstrating the effectiveness of our techniques.

We believe the proposed work can be extended in many interesting directions. Query dependent thinning is one important future work. When a query selects a subset of features, depending on its selectivity, the pre-computed thinning result can look sparse. Showing more features by relaxing the thinning result is desirable in such cases. We can also improve the algorithm's scalability even further by disk-based or distributed thinning algorithms. Another future direction is to apply our work on the continuous zoom case. One possible adaptation to the continuous case is to interpolate solutions between two discrete zoom levels, selecting a subset of features from the difference of features between the two levels. Another possibility is to define the problem with continuous zoom from the start, where cells in a hierarchy form a square pyramid in 3D space.

REFERENCES

AGMON, S. 1954. The relaxation method for linear inequalities. *Canadian Journal of Mathematics 5,* 3, 388–414.

BEEN, K., DAICHES, E., AND YAP, C.-K. 2006. Dynamic map labeling. *IEEE Transaction on Visualization and Computer Graphics 12,* 5, 773 – 780.

BEEN, K., NÖLLENBURG, M., POON, S.-H., AND WOLFFD, A. 2010. Optimizing active ranges for consistent dynamic map labeling. *Computational Geometry 43,* 3, 312 – 328.

CHAN, S., XIAO, L., GERTH, J., AND HANRAHAN, P. 2008. Atlas: Maintaining interactivity while exploring massive time series. In *IEEE Symposium on Visual Analytics Science and TEchnology (VAST '08)*. 59 – 66.

COCHRAN, W. G. 1977. *Sampling Techniques, 3rd Edition*. John Wiley.

COHEN, S., LI, C., YANG, J., AND YU, C. 2011. Computational journalism: A call to arms to database researchers. In *Proceedings of the Conference on Innovative Data Systems Research*. 148 – 151.

DAS SARMA, A., LEE, H., GONZALEZ, H., MADHAVAN, J., AND HALEVY, A. Y. 2012. Efficient spatial sampling of large geographical tables. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*. 193 – 204.

DIX, A. AND ELLIS, G. 2002. by chance enhancing interaction with large data sets through statistical sampling. In *Proceedings of the Working Conference on Advanced Visual Interfaces*. 167 – 176.

ESRI. 2012. Arcgis. http://www.esri.com/software/arcgis/index.html.

FISHER, D., POPO, I., DRUCKER, S. M., AND SCHRAEFEL, M. 2012. Trust me, i'm partially right: Incremental visualization lets analysts explore large datasets faster. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 1673 – 1682.

FRANK, A. U. AND TIMPF, S. 1994. Multiple representations for cartographic objects in a multi-scale tree - an intelligent graphical zoom. *Computers and Graphics 18,* 6, 823 – 829.

GAREY, M. R. AND JOHNSON, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co.

GEOIQ. 2012. Geocommons. `http://geocommons.com/`.

GONZALEZ, H., HALEVY, A. Y., JENSEN, C. S., LANGEN, A., MADHAVAN, J., SHAPLEY, R., SHEN, W., AND GOLDBERG-KIDON, J. 2010. Google fusion tables: web-centered data management and collaboration. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*. 1061–1066.

GOOGLE. 2005. Google maps. http://maps.google.com.

GRUMBACH, S., RIGAUX, P., AND SEGOUFIN, L. 1998. The dedale system for complex spatial queries. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*.

GUTTMAN, A. 1984. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*. 47–57.

HAN, J. AND KAMBER, M. 2000. *Data Mining: Concepts and Techniques*. Morgan Kaufmann.

HAN, J., KAMBER, M., AND TUNG, A. K. H. 2001. Spatial clustering methods in data mining: A survey. *Geographic Data Mining and Knowledge Discovery*, 1 – 29.

HILBERT, D. 1891. Uber die stetige abbildung einer linie auf ein flachenstuck. *Math. Ann. 38*, 459–460.

HJALTASON, G. R. AND SAMET, H. 1998. Incremental distance join algorithms for spatial databases. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*. 237–248.

ILYAS, I. F., BESKALES, G., AND SOLIMAN, M. A. 2008. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys 40,* 4, 11–58.

OLSON, D. M., DINERSTEIN, E., WIKRAMANAYAKE, E., BURGESS, N., POWELL, G., UNDERWOOD, E., D'AMICO, J., ITOUA, I., STRAND, H., MORRISON, J., LOUCKS, C., ALLNUTT, T., RICKETTS, T., KURA, Y., LAMOREUX, J., W.W.WETTENGEL, HEDAO, P., AND KASSEM, K. 2001. Terrestrial ecoregions of the world: A new map of life on earth. *BioScience 51*, 933–938.

ORACLE. 2007. Oracle spatial. `http://www.oracle.com/us/products/database/options/spatial/index.html`.

PATEL, J., YU, J., KABRA, N., TUFTE, K., NAG, B., BURGER, J., HALL, N., RAMASAMY, K., LUEDER, R., ELLMANN, C., KUPSCH, J., GUO, S., LARSON, J., DEWITT, D., AND NAUGHTON, J. 1997. Building a scalable geo-spatial dbms: Technology, implementation, and evaluation. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*. 336–347.

PETZOLD, I., GRÖGER, G., AND PLÜMER, L. 2003. Fast screen map labeling - data structures and algorithms. In *Proceedings of International Cartographic Conference*. 288 – 298.

PHILLIPS, R. AND NOYES, L. 1982. An investigation of visual clutter in the topographic base of a geological map. *Cartographic Journal 19,* 2, 122 – 131.

PIRINGER, H., TOMINSKI, C., MUIGG, P., AND BERGER, W. 2009. A multi-threading architecture to support interactive visual exploration. *IEEE Trans. on Visualization and Computer Graphics 15,* 6, 1113 – 1120.

PUPPO, E. AND DETTORI, G. 1995. Towards a formal model for multiresolution spatial maps. In *International Simposium on Large Spatial Database*. 152–169.

SAGAN, H. 1994. *Space-Filling Curves*. Springer-Verlag.

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

SCHRIJVER, A. 1986. *Theory of Linear and Integer Programming*. John Wiley.

SHEA, K. AND MCMASTER, R. 1989. Cartographic generalization in a digital environment: When and how to generalize. *AutoCarto 9*, 56–67.

STOLTE, C., TANG, D., AND HANRAHAN, P. 2003. Multiscale visualization using data cubes. *IEEE Transaction on Visualization and Computer Graphics 9,* 2, 176 – 187.

THOMAS, J. AND (EDS.), K. A. C. 2005. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. IEEE Computer Society Press.

TRYFONA, N. AND EGENHOFER, M. J. 1997. Consistency among parts and aggregates: A computation model. *Transactions in GIS 1,* 3, 189 – 206.

VIZZUALITY. 2012. Cartodb. http://cartodb.com.

WARE, M. J., JONES, C. B., AND THOMAS, N. 2003. Automated map generalization with multiple operators: a simulated annealing approach. *International Journal of Geographical Information Science 17,* 8, 743 – 769.

WOODRUFF, A., LANDAY, J., AND STONEBRAKER, M. 1998. Constant information density in zoomable interfaces. In *Proceedings of the Working Conference on Advanced Visual Interfaces*. 57–65.