Advanced Data Management (CSCI 640/490)

Spatial Data

Dr. David Koop
Data Exploration Through Visualization
Data Exploration Through Visualization
Why do we visualize data?

Figures are richer; provide more information with less clutter and in less space. Figures provide the gestalt effect: they give an overview; make structure more visible.

Figures are more accessible, easier to understand, faster to grasp, more comprehensible, more memorable, more fun, and less formal.

List adapted from: [Stasko et al. 1998]
## Why Visual?

<table>
<thead>
<tr>
<th></th>
<th>I</th>
<th>II</th>
<th>III</th>
<th>IV</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x</td>
<td>y</td>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>10.0</td>
<td>8.04</td>
<td>10.0</td>
<td>9.14</td>
<td>10.0</td>
</tr>
<tr>
<td>8.0</td>
<td>6.95</td>
<td>8.0</td>
<td>8.14</td>
<td>8.0</td>
</tr>
<tr>
<td>13.0</td>
<td>7.58</td>
<td>13.0</td>
<td>8.74</td>
<td>13.0</td>
</tr>
<tr>
<td>9.0</td>
<td>8.81</td>
<td>9.0</td>
<td>8.77</td>
<td>9.0</td>
</tr>
<tr>
<td>11.0</td>
<td>8.33</td>
<td>11.0</td>
<td>9.26</td>
<td>11.0</td>
</tr>
<tr>
<td>14.0</td>
<td>9.96</td>
<td>14.0</td>
<td>8.10</td>
<td>14.0</td>
</tr>
<tr>
<td>6.0</td>
<td>7.24</td>
<td>6.0</td>
<td>6.13</td>
<td>6.0</td>
</tr>
<tr>
<td>4.0</td>
<td>4.26</td>
<td>4.0</td>
<td>3.10</td>
<td>4.0</td>
</tr>
<tr>
<td>12.0</td>
<td>10.84</td>
<td>12.0</td>
<td>9.13</td>
<td>12.0</td>
</tr>
<tr>
<td>7.0</td>
<td>4.82</td>
<td>7.0</td>
<td>7.26</td>
<td>7.0</td>
</tr>
<tr>
<td>5.0</td>
<td>5.68</td>
<td>5.0</td>
<td>4.74</td>
<td>5.0</td>
</tr>
</tbody>
</table>

[F. J. Anscombe]
## Why Visual?

<table>
<thead>
<tr>
<th></th>
<th>I</th>
<th>II</th>
<th>III</th>
<th>IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>x</td>
<td>y</td>
<td>x</td>
</tr>
<tr>
<td>10.0</td>
<td>8.04</td>
<td>10.0</td>
<td>9.14</td>
<td>10.0</td>
</tr>
<tr>
<td>8.0</td>
<td>6.95</td>
<td>8.0</td>
<td>8.14</td>
<td>8.0</td>
</tr>
<tr>
<td>13.0</td>
<td>7.58</td>
<td>13.0</td>
<td>8.74</td>
<td>13.0</td>
</tr>
<tr>
<td>9.0</td>
<td>8.81</td>
<td>9.0</td>
<td>8.77</td>
<td>9.0</td>
</tr>
<tr>
<td>11.0</td>
<td>8.33</td>
<td>11.0</td>
<td>9.26</td>
<td>11.0</td>
</tr>
<tr>
<td>14.0</td>
<td>9.96</td>
<td>14.0</td>
<td>8.10</td>
<td>14.0</td>
</tr>
<tr>
<td>6.0</td>
<td>7.24</td>
<td>6.0</td>
<td>6.13</td>
<td>6.0</td>
</tr>
<tr>
<td>4.0</td>
<td>4.26</td>
<td>4.0</td>
<td>3.10</td>
<td>4.0</td>
</tr>
<tr>
<td>12.0</td>
<td>10.84</td>
<td>12.0</td>
<td>9.13</td>
<td>12.0</td>
</tr>
<tr>
<td>7.0</td>
<td>4.82</td>
<td>7.0</td>
<td>7.26</td>
<td>7.0</td>
</tr>
<tr>
<td>5.0</td>
<td>5.68</td>
<td>5.0</td>
<td>4.74</td>
<td>5.0</td>
</tr>
</tbody>
</table>

### Summary Statistics

- **Mean of x**: 9
- **Variance of x**: 11
- **Mean of y**: 7.50
- **Variance of y**: 4.122
- **Correlation**: 0.816
Why Visual?

[F. J. Anscombe]
Why Visual?

- **Mean of x**: 9
- **Variance of x**: 11
- **Mean of y**: 7.50
- **Variance of y**: 4.122
- **Correlation**: 0.816

---

[F. J. Anscombe]
Visual Pop-out
Supporting Scalable Visualization

- Two Problems:
  - **Lots of data**, how to display (encode) it
  - User **interaction** is key to gaining insight, requires **low latency**

- Addressing big data:
  - Encoding should focus on **available resolution**, not size of data
  - Approaches:
    - Sampling
    - Modeling
    - **Binning**
      - Bin → Aggregate (→ Smooth) → Plot
Time Series Aggregation

1M samples -> 2,653 plotted points
Time Series Aggregation

- Insight: the **resolution** is bound by the **number of pixels**
- Compute average value per pixel (1 point/pixel)
  - …this may miss extreme (min, max) values
- Plot min/max values per pixel (2 points/pixel)
  - …this does better, but still misrepresents
- M4: min/max values & timestamps (4 points/pixel)
  - …this provides provable fidelity to the full data!

[Jugel, 2014, via J. Heer]
Effects of Latency

• **Higher latency** leads to...
  - Reduced user activity and data set coverage
  - Significantly fewer brushing actions
  - Less observation, generalization & hypothesis

• **Interaction effect**: Exposure to delay reduces subsequent performance in low-latency interface.

• Different interactions exhibit **varied sensitivity** to latency. Brushing is highly sensitive!

[Liu et al. via J. Heer]
Interactive Scalability Solutions

- Use Database Technology: databases built for scalability
- Client-side Indexing (Data Cubes): take advantage of data structure
- Prefetching: load data before requests based on predictions
- Approximation: show estimates early but with error information
Sampling vs. Aggregation

Figure 1: A symbol map (a) and heatmap (b) visualizing a dataset of Brightkite user checkins. The symbol map visualizes a sample of the data, and the heatmap shows the density of checkins by aggregation. Compared to the heatmap, sampling misses important structures such as inter-state highway travel and Hurricane Ike, while dense regions still suffer from over-plotting.

**Sampling vs. Aggregation**

Sampling and aggregation are two approaches used to handle large datasets. Sampling involves selecting a subset of the data to analyze, which can be useful for preliminary exploration but may not capture all the nuances of the full dataset. Aggregation, on the other hand, involves combining data from multiple sources into a single representation, which can provide a more comprehensive overview. Both methods have their advantages and disadvantages, and the choice depends on the specific requirements of the analysis.

In the context of visual analytics, sampling and aggregation are often used in combination. For instance, in the visualization of the Brightkite dataset, a symbol map provides a detailed view of individual checkins, while a heatmap aggregates the data to show density patterns. This hybrid approach allows users to explore both detailed and aggregate information simultaneously.

**Binned Aggregation**

Binning aggregates data and visualizes density by counting the number of data points falling within each predefined bin. This method is particularly useful for large datasets, as it can summarize the data into a more manageable form while preserving important trends. Binning can also be used to define aggregates at multiple scales over a hierarchy, allowing for flexible exploration of the data.

**Model-based Abstraction**

Model-based abstraction involves using mathematical models or statistical summaries to represent data. For example, moving averages and auto-regressive models can be used to fit trend lines, providing insights into temporal patterns. This approach is valuable for time series data, where understanding the underlying trends is crucial.

**Hybrid Reduction Methods**

Hybrid reduction methods combine sampling and aggregation to provide fast approximate answers, often through multi-scale histograms of network statistics. This technique is particularly useful for interactive visual analytics, where real-time responsiveness is essential.

**Visual Summation**

To visually summarize data, techniques such as heatmaps, box plots, and other statistical visualizations are employed. Heatmaps, for example, are exemplary 1D and 2D binned plots that can aggregate values over a continuous range. Categorical variables can also be treated as bins for aggregative purposes.

**Reduction Strategies**

Another reduction strategy is to describe data in terms of specific data outliers along with the bins. This approach helps in identifying unusual patterns that might not be apparent through simple aggregation. For instance, a box plot with outliers applies both modeling and filtering to highlight significant deviations from the norm.

**References**

- Zhicheng Liu, Biye Jiang & Jeffrey Heer / imMens: Real-time Visual Querying of Big Data
- © 2013 The Author(s)
- © 2013 The Eurographics Association and Blackwell Publishing Ltd.
Full 5-D Data Cube

[Month, Day, Hour, X, Y]
~2.3 Billion Bins
Break into Tiles

[Image - Break into Tiles]

- **Month0-11-0**: Bar chart showing the distribution of checkins by month.
- **Day0-30-0**: Bar chart showing the distribution of checkins by day.
- **Hour0-23-0**: Bar chart showing the distribution of checkins by hour.

[Z. Liu et al., 2013]
Data Cube Decomposition

~2.3 B Bins

~17.6 M Bins

3-D cubes

3-D data tiles

[Z. Liu et al., 2013]
Figure 5: (a) A 5-dimensional data cube of Brightkite check-ins; (b) Decomposing a full cube into sub-cubes and data tiles.

dimension as $Db_s - b_e - z$, where $D$ is the binned data dimension, $b_s$ represents the starting bin index, $b_e$ represents the ending bin index, and $z$ represents the zoom level.

The Brightkite visualization in Figure 4 uses 13 data tiles: one tile representing the 3-dimensional projection of month, day and hour (i.e., $Month0 - 11 - 0 \times Day0 - 30 - 0 \times Hour0 - 23 - 0$), and twelve tiles containing 3-dimensional projections for all combinations of the four geographic segments and three histograms (e.g., $X256 - 511 - 4 \times Y512 - 767 - 4 \times Month0 - 11 - 0$).

Multivariate data tiles are precomputed on a server and then loaded on demand to support client-side visualization. Brushing & linking involves aggregating these data tiles. For example, when the user selects a region in the geographic heatmap, we need to highlight the corresponding checkin distributions in the three histograms. In the worst case, the selected geographic region covers bins in all four map tiles. To render the highlight in the $Day$ histogram we need to roll-up the four data tiles containing the $X \times Y \times Day$ dimensions and sum the results. Figure 6 shows this process.

For interactions like panning and zooming, we dynamically fetch data tiles precomputed at different levels of binning resolution, similar to existing mapping services.

5.3. Dense vs. Sparse Data Tile Storage

Data tiles can use either sparse or dense packing schemes. A sparse representation stores indices and values only for non-zero bins (Figure 7 (b)). A dense representation includes zero values, but can store all the data as a simple array if the bin counts for all dimensions are known (Figure 7 (c)).

If a data tile has many empty bins, a sparse representation can reduce storage costs. For example, a sparse packing is used in Profiler [KPP $\times 12$] for full data cubes of up to 5 dimensions.

We treat row indices as numbers in a mixed-radix number system [Knu06]. The row index in a $k$-dimensional data tile can be expressed as $V(k_1)R(k_1) | V(k_2)R(k_2) | ... | V(0)R(0)$, where $V(k)$ is the value of the $k$th digit, and $R(k)$ is the radix of the $k$th digit.

Figure 6: Brushing & linking from the geographic heatmap to the $Day$ histogram. We aggregate four data tiles along the $X$ and $Y$ dimensions and sum up the projections.

Figure 7: Sparse and dense representations of a data tile.

However, as the number of data records increases, the density of the data typically also increases. Once the proportion of non-zero bins exceeds a threshold (20% for 4D tiles, 25% for 3D tiles), a dense representation is more efficient because we can omit bin indices. In imMens we use dense tiles to exploit these space savings, safeguard worst-case performance, and simplify parallel query processing.

5.4. Parallel Query Processing

A dense representation scheme supports simple, efficient parallel processing when aggregating data tiles. Dense tiles provide a consistent indexing scheme that enables direct
Prefetching (ForeCache)

- Predict which tiles a user will need next and prefetch those
  - Use common patterns (zoom, pan)
  - Use regions of interest (ROIs)

![Map with ROI tiles highlighted](image)

[Battle et al., 2016]
Latency Differences in Tasks

Brushing is more common and people are sensitive to latencies.

Prioritize brushing latency over view switching latency.

[D. Moritz et al. via J. Heer]
Task-Prioritized Prefetching

brings in the precomputed view
serves requests from a data cube
interacts with a new view
query for new data cubes
Assignment 5

- Spatial, Graph, and Temporal Data Processing
Data Cubes

J. Han, M. Kamber, and J. Pei
Data Cube: A Lattice of Cuboids

0-D (apex) cuboid

1-D cuboids

2-D cuboids

3-D cuboids

4-D (base) cuboid

[Han et al., 2011]
Cube Operations

- Roll-up: aggregate up the given hierarchy
- Drill-down: refine down the given hierarchy
- Roll-up and drill-down are "inverses"
Spatial Data Exploration Motivation

L. Battle
Nanocubes for Real-Time Exploration of Spatiotemporal Datasets

L. Lins, J. T. Klosowski, and C. Scheidegger
Goal: Interactive Exploration of Data Cubes

Linked view of tweets in San Diego, US

[Lins et al., 2013]
Fig. 1. Example visualizations of 210 million public geolocated Twitter posts over the course of a year. The data structure we propose enables real-time (these images above were rendered faster than the typical screen refresh rate) visual exploration of large, spatiotemporal, multidimensional datasets. The visual encodings built using nanocubes are within a controllable difference to ones rendered by a traditional linear scan over the dataset. They naturally support linked navigation and brushing, and include choropleth maps, time series over arbitrary regions and scales of space and time, parallel sets, histograms, and binned scatterplots. The color scale of the choropleth map is a diverging scale in which blue corresponds to iPhones being relatively more popular, and red corresponds to higher relative popularity of Android devices.

Abstract

—Consider real-time exploration of large multidimensional spatiotemporal datasets with billions of entries, each defined by a location, a time, and other attributes. Are certain attributes correlated spatially or temporally? Are there trends or outliers in the data? Answering these questions requires aggregation over arbitrary regions of the domain and attributes of the data. Many relational databases implement the well-known data cube aggregation operation, which in a sense precomputes every possible aggregate query over the database. Data cubes are sometimes assumed to take a prohibitively large amount of space, and to consequently require disk storage. In contrast, we show how to construct a data cube that fits in a modern laptop’s main memory, even for billions of entries; we call this data structure a nanocube. We present algorithms to compute and query a nanocube, and show how it can be used to generate well-known visual encodings such as heatmaps, histograms, and parallel coordinate plots. When compared to exact visualizations created by scanning an entire dataset, nanocube plots have bounded screen error across a variety of scales, thanks to a hierarchical structure in space and time. We demonstrate the effectiveness of our technique on a variety of real-world datasets, and present memory, timing, and network bandwidth measurements. We find that the timings for the queries in our examples are dominated by network and user-interaction latencies.

Index Terms

—Data cube, Data structures, Interactive exploration

1INTRODUCTION

As datasets get larger, exploratory data visualization becomes more difficult. Consider a dataset with a billion entries. We can compute a small summary of the dataset and visualize the summary instead of the dataset, but as Anscombe’s famous quartet shows [3], summaries themselves cannot ascertain their own validity. Summaries might help, but in order to understand if that is the case, we will inevitably find ourselves having to visualize one billion residuals. As far as scale goes, we are back to square one. In other words, data summarization alone will never solve the problem of scale in exploratory visualization. As visualization practitioners, what then can we do? Even drawing the simplest scatterplot is not straightforward. If we decide to produce the visualization by scanning the rows of a table, we will either need non-trivial parallel rendering algorithms or significant time to produce a drawing. Neither of these solutions is attractive or scales well with dataset size.

Data cubes are structures that perform aggregations across every possible set of dimensions of a table in a database, to support quick exploration [15,31]. Many visualization systems are built on top of data cubes, concretely or conceptually. Still, only recently have researchers started to examine data cube creation algorithms in the context of information visualization [33,18,21].

Data cubes are often problematic in that they can take prohibitively large amounts of memory as the number of dimensions increases. In
Nanocubes for Real-Time Exploration of Spatiotemporal Datasets

Lauro Lins, James T. Klosowski, and Carlos Scheidegger

Fig. 1. Example visualizations of 210 million public geolocated Twitter posts over the course of a year. The data structure we propose enables real-time (these images above were rendered faster than the typical screen refresh rate) visual exploration of large, spatiotemporal, multidimensional datasets. The visual encodings built using nanocubes are within a controllable difference to ones rendered by a traditional linear scan over the dataset. They naturally support linked navigation and brushing, and include choropleth maps, time series over arbitrary regions and scales of space and time, parallel sets, histograms, and binned scatterplots. The color scale of the choropleth map is a diverging scale in which blue corresponds to iPhones being relatively more popular, and red corresponds to higher relative popularity of Android devices.

Abstract

—Consider real-time exploration of large multidimensional spatiotemporal datasets with billions of entries, each defined by a location, a time, and other attributes. Are certain attributes correlated spatially or temporally? Are there trends or outliers in the data? Answering these questions requires aggregation over arbitrary regions of the domain and attributes of the data. Many relational databases implement the well-known data cube aggregation operation, which in a sense precomputes every possible aggregate query over the database. Data cubes are sometimes assumed to take a prohibitively large amount of space, and to consequently require disk storage. In contrast, we show how to construct a data cube that fits in a modern laptop’s main memory, even for billions of entries; we call this data structure a nanocube. We present algorithms to compute and query a nanocube, and show how it can be used to generate well-known visual encodings such as heatmaps, histograms, and parallel coordinate plots. When compared to exact visualizations created by scanning an entire dataset, nanocube plots have bounded screen error across a variety of scales, thanks to a hierarchical structure in space and time. We demonstrate the effectiveness of our technique on a variety of real-world datasets, and present memory, timing, and network bandwidth measurements. We find that the timings for the queries in our examples are dominated by network and user-interaction latencies.

Index Terms — Data cube, Data structures, Interactive exploration

1 INTRODUCTION

As datasets get larger, exploratory data visualization becomes more difficult. Consider a dataset with a billion entries. We can compute a small summary of the dataset and visualize the summary instead of the dataset, but as Anscombe’s famous quartet shows[3], summaries themselves cannot ascertain their own validity. Summaries might help, but in order to understand if that is the case, we will inevitably find ourselves having to visualize one billion residuals. As far as scale goes, we are back to square one. In other words, data summarization alone will never solve the problem of scale in exploratory visualization. As visualization practitioners, what then can we do? Even drawing the simplest scatterplot is not straightforward. If we decide to produce the visualization by scanning the rows of a table, we will either need non-trivial parallel rendering algorithms or significant time to produce a drawing. Neither of these solutions is attractive or scales well with dataset size.

Data cubes are structures that perform aggregations across every possible set of dimensions of a table in a database, to support quick exploration[15, 31]. Many visualization systems are built on top of data cubes, concretely or conceptually. Still, only recently have researchers started to examine data cube creation algorithms in the context of information visualization[33, 18, 21]. Data cubes are often problematic in that they can take prohibitively large amounts of memory as the number of dimensions increases. In

iPhone vs. Android Map

[Image of iPhone vs. Android Map] Lins et. al, 2013
Nanocubes for Real-Time Exploration of Spatiotemporal Datasets

Lauro Lins, James T. Klosowski, and Carlos Scheidegger

Fig. 1. Example visualizations of 210 million public geolocated Twitter posts over the course of a year. The data structure we propose enables real-time (these images above were rendered faster than the typical screen refresh rate) visual exploration of large, spatiotemporal, multidimensional datasets. The visual encodings built using nanocubes are within a controllable difference to ones rendered by a traditional linear scan over the dataset. They naturally support linked navigation and brushing, and include choropleth maps, time series over arbitrary regions and scales of space and time, parallel sets, histograms, and binned scatterplots. The color scale of the choropleth map is a diverging scale in which blue corresponds to iPhones being relatively more popular, and red corresponds to higher relative popularity of Android devices.

Abstract

—Consider real-time exploration of large multidimensional spatiotemporal datasets with billions of entries, each defined by a location, a time, and other attributes. Are certain attributes correlated spatially or temporally? Are there trends or outliers in the data? Answering these questions requires aggregation over arbitrary regions of the domain and attributes of the data. Many relational databases implement the well-known data cube aggregation operation, which in a sense precomputes every possible aggregate query over the database. Data cubes are sometimes assumed to take a prohibitively large amount of space, and to consequently require disk storage. In contrast, we show how to construct a data cube that fits in a modern laptop’s main memory, even for billions of entries; we call this data structure a nanocube. We present algorithms to compute and query a nanocube, and show how it can be used to generate well-known visual encodings such as heatmaps, histograms, and parallel coordinate plots. When compared to exact visualizations created by scanning an entire dataset, nanocube plots have bounded screen error across a variety of scales, thanks to a hierarchical structure in space and time. We demonstrate the effectiveness of our technique on a variety of real-world datasets, and present memory, timing, and network bandwidth measurements. We find that the timings for the queries in our examples are dominated by network and user-interaction latencies.

Index Terms

—Data cube, Data structures, Interactive exploration

1INTRODUCTION

As datasets get larger, exploratory data visualization becomes more difficult. Consider a dataset with a billion entries. We can compute a small summary of the dataset and visualize the summary instead of the dataset, but as Anscombe’s famous quartet shows, summaries themselves cannot ascertain their own validity. Summaries might help, but in order to understand if that is the case, we will inevitably find ourselves having to visualize one billion residuals. As far as scale goes, we are back to square one. In other words, data summarization alone will never solve the problem of scale in exploratory visualization. As visualization practitioners, what then can we do? Even drawing the simplest scatterplot is not straightforward. If we decide to produce the visualization by scanning the rows of a table, we will either need non-trivial parallel rendering algorithms or significant time to produce a drawing. Neither of these solutions is attractive or scales well with dataset size.

Data cubes are structures that perform aggregations across every possible set of dimensions of a table in a database, to support quick exploration. Many visualization systems are built on top of data cubes, concretely or conceptually. Still, only recently have researchers started to examine data cube creation algorithms in the context of information visualization.

Data cubes are often problematic in that they can take prohibitively large amounts of memory as the number of dimensions increases. In our approach, we take advantage of the fact that only a few of the possible combinations of dimensions are commonly needed. In particular, we can construct a data structure that captures these combinations, and is stored in memory.

Zoom into Chicago

[Fig. 1. Example visualizations of 210 million public geolocated Twitter posts over the course of a year. The data structure we propose enables real-time (these images above were rendered faster than the typical screen refresh rate) visual exploration of large, spatiotemporal, multidimensional datasets. The visual encodings built using nanocubes are within a controllable difference to ones rendered by a traditional linear scan over the dataset. They naturally support linked navigation and brushing, and include choropleth maps, time series over arbitrary regions and scales of space and time, parallel sets, histograms, and binned scatterplots. The color scale of the choropleth map is a diverging scale in which blue corresponds to iPhones being relatively more popular, and red corresponds to higher relative popularity of Android devices.]

[Fig. 2. A zoomed-in view of the Chicago area, showing a heatmap of iPhone versus Android usage. The colors indicate the relative popularity of each phone type in different regions.]

[Lins et. al, 2013]
Nanocubes for Real-Time Exploration of Spatiotemporal Datasets

Lauro Lins, James T. Klosowski, and Carlos Scheidegger

Fig. 1. Example visualizations of 210 million public geolocated Twitter posts over the course of a year. The data structure we propose enables real-time (these images above were rendered faster than the typical screen refresh rate) visual exploration of large, spatiotemporal, multidimensional datasets. The visual encodings built using nanocubes are within a controllable difference to ones rendered by a traditional linear scan over the dataset. They naturally support linked navigation and brushing, and include choropleth maps, time series over arbitrary regions and scales of space and time, parallel sets, histograms, and binned scatterplots. The color scale of the choropleth map is a diverging scale in which blue corresponds to iPhones being relatively more popular, and red corresponds to higher relative popularity of Android devices.

Abstract

—Consider real-time exploration of large multidimensional spatiotemporal datasets with billions of entries, each defined by a location, a time, and other attributes. Are certain attributes correlated spatially or temporally? Are there trends or outliers in the data? Answering these questions requires aggregation over arbitrary regions of the domain and attributes of the data. Many relational databases implement the well-known data cube aggregation operation, which in a sense precomputes every possible aggregate query over the database. Data cubes are sometimes assumed to take a prohibitively large amount of space, and to consequently require disk storage. In contrast, we show how to construct a data cube that fits in a modern laptop's main memory, even for billions of entries; we call this data structure a nanocube. We present algorithms to compute and query a nanocube, and show how it can be used to generate well-known visual encodings such as heatmaps, histograms, and parallel coordinate plots. When compared to exact visualizations created by scanning an entire dataset, nanocube plots have bounded screen error across a variety of scales, thanks to a hierarchical structure in space and time. We demonstrate the effectiveness of our technique on a variety of real-world datasets, and present memory, timing, and network bandwidth measurements. We find that the timings for the queries in our examples are dominated by network and user-interaction latencies.

Index Terms
—Data cube, Data structures, Interactive exploration

1INTRODUCTION

As datasets get larger, exploratory data visualization becomes more difficult. Consider a dataset with a billion entries. We can compute a small summary of the dataset and visualize the summary instead of the dataset, but as Anscombe's famous quartet shows, summaries themselves cannot ascertain their own validity. Summaries might help, but in order to understand if that is the case, we will inevitably find ourselves having to visualize one billion residuals. As far as scale goes, we are back to square one. In other words, data summarization alone will never solve the problem of scale in exploratory visualization. As visualization practitioners, what then can we do? Even drawing the simplest scatterplot is not straightforward. If we decide to produce the visualization by scanning the rows of a table, we will either need non-trivial parallel rendering algorithms or significant time to produce a drawing. Neither of these solutions is attractive or scales well with dataset size.

Data cubes are structures that perform aggregations across every possible set of dimensions of a table in a database, to support quick exploration. Many visualization systems are built on top of data cubes, concretely or conceptually. Still, only recently have researchers started to examine data cube creation algorithms in the context of information visualization. Data cubes are often problematic in that they can take prohibitively large amounts of memory as the number of dimensions increases. In

SuperBowl in Indianapolis

Total count: 1,686,897 of 210,634,624

device
windows
ipad
android
iphone
none

[SuperBowl in Indianapolis, 2012, data visualization showing device usage and timeline]
New Year's Eve in Manhattan

Device distribution for New Year's Eve in Manhattan:
- Windows
- iPad
- Android
- iPhone
- None

Total count: 2,052,349 of 210,634,624

[Fig. 1. Example visualizations of 210 million public geolocated Twitter posts over the course of a year. The data structure we propose enables real-time (these images above were rendered faster than the typical screen refresh rate) visual exploration of large, spatiotemporal, multidimensional datasets. The visual encodings built using nanocubes are within a controllable difference to ones rendered by a traditional linear scan over the dataset. They naturally support linked navigation and brushing, and include choropleth maps, time series over arbitrary regions and scales of space and time, parallel sets, histograms, and binned scatterplots. The color scale of the choropleth map is a diverging scale in which blue corresponds to iPhones being relatively more popular, and red corresponds to higher relative popularity of Android devices.]
Aggregations on Spatiotemporal Data

• Spatial: e.g. counting events in a spatial region (world or San Fran.)
• Temporal: e.g. queries at multiple scares (hour, day, week, month)
• Seek to address Visual Information Seeking Mantra:
  • Overview first, zoom and filter, details-on-demand
• Multidimensional:
  - Latitude, Longitude, Time + more
Data Cube Aggregations

Relation A

<table>
<thead>
<tr>
<th>Country</th>
<th>Device</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>US</td>
<td>Android</td>
<td>en</td>
</tr>
<tr>
<td>US</td>
<td>iPhone</td>
<td>ru</td>
</tr>
<tr>
<td>South Africa</td>
<td>iPhone</td>
<td>en</td>
</tr>
<tr>
<td>India</td>
<td>Android</td>
<td>en</td>
</tr>
<tr>
<td>Australia</td>
<td>iPhone</td>
<td>en</td>
</tr>
</tbody>
</table>

Aggregation B

<table>
<thead>
<tr>
<th>Country</th>
<th>Device</th>
<th>Language</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>All</td>
<td>All</td>
<td>5</td>
</tr>
</tbody>
</table>

Group By on Device, Language C

<table>
<thead>
<tr>
<th>Country</th>
<th>Device</th>
<th>Language</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>Android</td>
<td>en</td>
<td>2</td>
</tr>
<tr>
<td>All</td>
<td>iPhone</td>
<td>en</td>
<td>2</td>
</tr>
<tr>
<td>All</td>
<td>iPhone</td>
<td>ru</td>
<td>1</td>
</tr>
</tbody>
</table>

Cube on Device, Language D

<table>
<thead>
<tr>
<th>Country</th>
<th>Device</th>
<th>Language</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>All</td>
<td>All</td>
<td>5</td>
</tr>
<tr>
<td>All</td>
<td>Android</td>
<td>All</td>
<td>2</td>
</tr>
<tr>
<td>All</td>
<td>iPhone</td>
<td>All</td>
<td>3</td>
</tr>
<tr>
<td>All</td>
<td>All</td>
<td>en</td>
<td>4</td>
</tr>
<tr>
<td>All</td>
<td>All</td>
<td>ru</td>
<td>1</td>
</tr>
<tr>
<td>All</td>
<td>iPhone</td>
<td>ru</td>
<td>1</td>
</tr>
<tr>
<td>All</td>
<td>Android</td>
<td>en</td>
<td>2</td>
</tr>
<tr>
<td>All</td>
<td>iPhone</td>
<td>en</td>
<td>2</td>
</tr>
</tbody>
</table>

Equivalent to Group By on all possible subsets of \{Device, Language\}

[Lins et. al, 2013]
Nanocube Queries

- Representing natural language queries as data cube queries

<table>
<thead>
<tr>
<th>Natural language query</th>
<th>s</th>
<th>c</th>
<th>t</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>count of all Delta flights</td>
<td>$R$</td>
<td>$U$</td>
<td>$R$</td>
<td>${\text{Delta}}$</td>
</tr>
<tr>
<td>count of all Delta flights in the Midwest</td>
<td>$R$</td>
<td>Midwest</td>
<td>$R$</td>
<td>${\text{Delta}}$</td>
</tr>
<tr>
<td>count of all flights in 2010</td>
<td>$R$</td>
<td>$U$</td>
<td></td>
<td>$2010$</td>
</tr>
<tr>
<td>time-series of all United flights in 2009</td>
<td>$R$</td>
<td>$U$</td>
<td>$D$</td>
<td>$2009$</td>
</tr>
<tr>
<td>heatmap of Delta flights in 2010</td>
<td>$D$</td>
<td>tile0</td>
<td>$R$</td>
<td>${\text{Delta}}$</td>
</tr>
</tbody>
</table>

- $s = \text{space}, \ c = \text{category}, \ t = \text{time}$
- $R = \text{rollup}, \ D = \text{drill down}$
- $<\text{value}>$ after $RD = \text{subset of dimension's domain}, \ U = \text{universe}$
- Note that time queries are stored in an array of cumulative counts

[Refs: Lins et. al, 2013]
Building a Nanocube

Indexing Schema

\[ S = [\ell_{\text{spatial1}}, \ell_{\text{spatial2}}, \ell_{\text{device}}] \]
Summed-area Table

- Every node in the previous figure stores an array of timestamped counts like this:

```
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td></td>
<td></td>
<td></td>
<td>**</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

query/tseries/1/3/4

start at bin 1, use buckets of 3 bins each, and collect 4 of these buckets

solve using...

```
<table>
<thead>
<tr>
<th>bin: 1</th>
<th>bin: 3</th>
<th>bin: 4</th>
<th>bin: 6</th>
<th>bin: 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>accum: 2</td>
<td>accum: 3</td>
<td>accum: 4</td>
<td>accum: 7</td>
<td>accum: 9</td>
</tr>
</tbody>
</table>
```

A Summed Table Sparse Representation for Counts

result

```
3 4 0 2
```

[Lins et. al, 2013]
Building a Nanocube: Step 1

Indexing Schema

\[ S = [[\ell_{\text{spatial1}}, \ell_{\text{spatial2}}], [\ell_{\text{device}}]] \]

[Five Tweets: Location and Device]

\( \ell_{\text{device}}(O) = \text{Android} \)
\( \ell_{\text{device}}(\bullet) = \text{iPhone} \)

\( \ell_{\text{spatial1}} \)

\begin{tabular}{|c|c|c|c|}
\hline
0.1 & 1.1 & 0.0 & 1.0 \\
\hline
\end{tabular}

\( \ell_{\text{spatial2}} \)

\begin{tabular}{|c|c|c|c|}
\hline
00.11 & 01.11 & 10.11 & 11.11 \\
00.10 & 01.10 & 10.10 & 11.10 \\
00.01 & 01.01 & 10.01 & 11.01 \\
00.00 & 01.00 & 10.00 & 11.00 \\
\hline
\end{tabular}

parent-child (same dimension):

proper \quad \text{shared}

content (next dimension):

proper \quad \text{shared}

updated in current step

dimension boundary

[Lins et. al, 2013]
Building a Nanocube: Step 2

Indexing Schema

\[ S = \{ [\ell_{\text{spatial1}}, \ell_{\text{spatial2}}], [\ell_{\text{device}}] \} \]

Five Tweets: Location and Device

parent-child (same dimension):

proper
shared

content (next dimension):

proper
shared

updated in
current step

dimension
boundary

2.

[Sismanis et al., 2013]
Building a Nanocube: Step 3

3.

parent-child (same dimension):
proper
shared
content (next dimension):
proper
shared
updated in current step
dimension boundary

Five Tweets: Location and Device

Indexing Schema
S = [\{spatial1, spatial2\}, \{device\}]

[Lins et. al, 2013]
Building a Nanocube: Step 4

By

D. Koop, CSCI 640/490, Spring 2024

[Image 1337x128 to 1578x205]

[Lins et. al, 2013]
Building a Nanocube: Step 5

Lins et al., 2013
Nanocubes Discussion

- Save space by organizing the data in a manner that takes advantage of data sparseness
- Limited to one spatial dimension, one temporal dimension
- Precompute once, then exploration has low latency
Example: American vs. Delta

[Image of map and data chart showing American Airlines and Delta Airlines' flight hotspots and temporal trends over time.]
Example: Cell Data Records

Fig. 10. Highlights of a visual analysis session of the CDR dataset, with 1,043,884,027 records. We noticed the different patterns in call volume by interacting with the dataset and trying different regions and category selections. Notice the patterns occur at different spatial and temporal scales.

In the following sections, we provide a brief overview of each of the datasets, followed by an overall summary of our experimental results in section 6.8. For each of the experiments, we paid particular attention to how much memory was required to build and store the nanocube index, as well as the overall complexity of the dataset itself, which varied greatly from one to the next. Once the nanocubes were constructed, we queried them using one or both of our front-end clients to highlight the ease with which analysts could explore the data.

The query times and bandwidth usage across all experiments are consistent, so we report them in aggregate here. The mean query time was 800 µs (less than 1 millisecond) with a maximum of 12 milliseconds. The output size per query averaged 5KB, with a maximum size of 50KB (geographical tiles dominated bandwidth usage). Our server currently uses no compression, although we plan to support transparent gzip stream encoding. The mean number of queries for the C++ client was 100 requests per second. The HTML5 client is much quieter, at around 1 query per second, since linked views are only updated when a brush is released. The C++ client was designed for LANs, and its bandwidth usage is around 5Mbps, well within current capacities.

6.1 Twitter
Between November 2011 and June 2012, we collected about 210 million tweets that originated in the United States using Twitter’s public feed which provides a representative sampling of all tweets. The rate of tweets obtained averaged about one million per day. The data was streamed in the form of JSON objects, from which we extracted the following attributes: latitude and longitude of the device, the time the tweet occurred, the client application used, the type of device, and the language of the tweet. The categorical dimensions in our data (application, device, language) had respectively 4, 5, and 15 distinct values. With a nanocube built using this data, we could quickly explore the data to better understand the areas in which one device is more popular than another, where each of the languages is most prevalent, and how that information changes over time (see Figure 7).

6.2 Airline Commercial Flights History
This publicly available dataset contains data for every commercial flight in the United States over a 20 year period (1987-2008) [2, 36]. For over 120 million flights, the records include the scheduled departure and arrival times, the actual departure and arrival times, the origin and destination airports, the airline, and other fields. For this experiment, we built our index using the origin airport (for latitude and longitude), scheduled departure time, the departure delay, and the airline. This allows us to answer queries related to overall departure delays for any airports, airlines, time of day, or combinations thereof. In Figure 8 we present an overview on the weekly percentages of total commercial flights in the U.S. for a 20 year period of Delta and American Airlines.

[Lins et. al, 2013]
Fig. 10. Highlights of a visual analysis session of the CDR dataset, with 1,043,884,027 records. We noticed the different patterns in call volume by interacting with the dataset and trying different regions and category selections. Notice the patterns occur at different spatial and temporal scales.

In the following sections, we provide a brief overview of each of the datasets, followed by an overall summary of our experimental results in section 6.8. For each of the experiments, we paid particular attention to how much memory was required to build and store the nanocube index, as well as the overall complexity of the dataset itself, which varied greatly from one to the next. Once the nanocubes were constructed, we queried them using one or both of our front-end clients to highlight the ease with which analysts could explore the data.

The query times and bandwidth usage across all experiments are consistent, so we report them in aggregate here. The mean query time was 800 µs (less than 1 millisecond) with a maximum of 12 milliseconds. The output size per query averaged 5KB, with a maximum size of 50KB (geographical tiles dominated bandwidth usage). Our server currently uses no compression, although we plan to support transparent gzip stream encoding. The mean number of queries for the C++ client was 100 requests per second. The HTML5 client is much quieter, at around 1 query per second, since linked views are only updated when a brush is released. The C++ client was designed for LANs, and its bandwidth usage is around 5Mbps, well within current capacities.

6.1 Twitter
Between November 2011 and June 2012, we collected about 210 million tweets that originated in the United States using Twitter's public feed which provides a representative sampling of all tweets. The rate of tweets obtained averaged about one million per day. The data was streamed in the form of JSON objects, from which we extracted the following attributes: latitude and longitude of the device, the time the tweet occurred, the client application used, the type of device, and the language of the tweet. The categorical dimensions in our data (application, device, language) had respectively 4, 5, and 15 distinct values. With a nanocube built using this data, we could quickly explore the data to better understand the areas in which one device is more popular than another, where each of the languages is most prevalent, and how that information changes over time (see Figure 7).

6.2 Airline Commercial Flights History
This publicly available dataset contains data for every commercial flight in the United States over a 20 year period (1987-2008). For over 120 million flights, the records include the scheduled departure and arrival times, the actual departure and arrival times, the origin and destination airports, the airline, and other fields. For this experiment, we built our index using the origin airport (for latitude and longitude), scheduled departure time, the departure delay, and the airline. This allows us to answer queries related to overall departure delays for any airports, airlines, time of day, or combinations thereof. In Figure 8 we present an overview on the weekly percentages of total commercial flights in the U.S. for a 20 year period of Delta and American Airlines.
Big Spatial Data Management

A. Eldawy