UCR-STAR: The UCR Spatio-temporal Active Repository

https://star.cs.ucr.edu/

Saheli Ghosh Tin Vu Mehrad Amin Eskandari Ahmed Eldawy
Department of Computer Science and Engineering, University of California, Riverside, USA
{sghos006,tvu032,mamin021,eldawy}@ucr.edu

Abstract

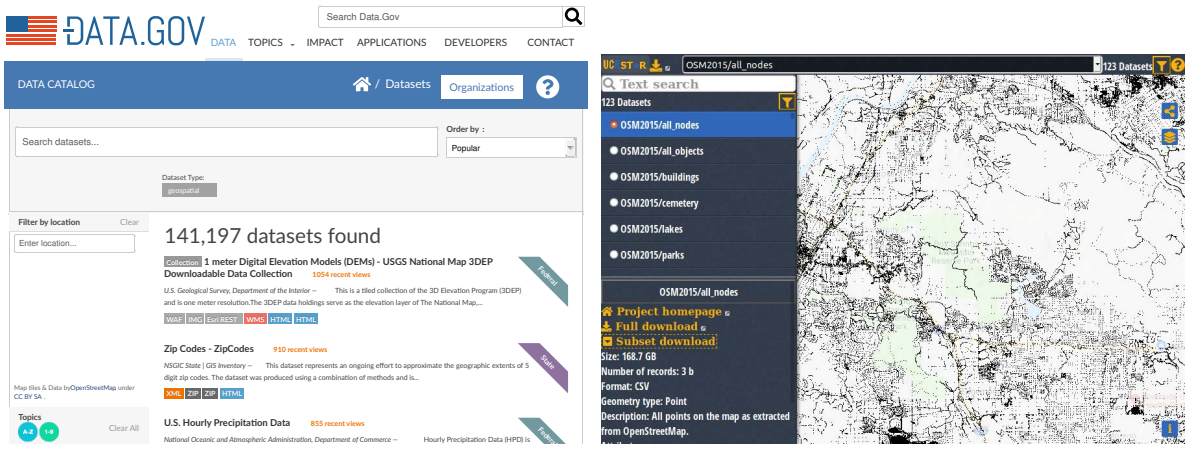
This article describes the UCR Spatio-temporal Active Repository (UCR-STAR). UCR-STAR is a visual catalog for big spatial datasets. Rather than a boring tabular listing of datasets, it provides an interactive map interface that allows users to explore these datasets to assess their coverage, quality, and distribution. This article describes both the functionality of UCR-STAR as well as the underlying system architecture. We believe that this article can help the research community by explaining how to realize research ideas into a workable product.

1 Introduction

Recently, there has been a tremendous growth in spatial data collection from various sources such as satellites, IoT sensors, smartphones, autonomous cars, and others. At the same time, there is a move for open data led by governments, non-profit organizations, and industry which makes hundreds of thousands of datasets publicly available. For example, Data.gov [4], which is maintained by the US government, hosts more than 140,000 datasets that are tagged as *geospatial*. Similarly, other governments, non-government organizations (NGOs), and companies keep releasing data on the web as part of the open data movement [3, 5, 13, 6, 11]. To browse these datasets, existing data repositories provide a plain listing of the datasets with references on how to access and download them, e.g., see Figure 1(a). Users will either have to *guess* what the dataset really contains or download these datasets, figure out how to import them into their favorite tool, before they can interact with the data.

To break from this old interface, this article describes UCR-STAR which provides an interactive web-based interface that allows users to interact with the datasets to assess their coverage, quality, and distribution before even downloading them. Figure 1(b) shows a screenshot of how UCR-STAR looks like. The majority of the screen is devoted to the map-based visualization of the dataset while a smaller part is used to list the datasets and its details. The map portion provides the standard map interactions such as pan and zoom. When a dataset is selected on the left, the map is updated in a fraction of a second to visualize the selected dataset no matter how big it is. Users can also search for datasets using a keyword search or advanced search based on the size or type of the dataset.

UCR-STAR is currently hosting more than 100 datasets with a total size of nearly one terabyte. We welcome requests to add additional datasets to the archive to further facilitate the access to these datasets and help the research community. The rest of this article gives more details about the architecture of UCR-STAR and the research behind it.



(a) Data.gov uses a plain listing of more than 140,000 datasets (b) The interactive map-based interface of UCR-STAR

Figure 1: A contrast between the plain static interface of Data.gov and the interactive interface of UCR-STAR

2 UCR-STAR Architecture

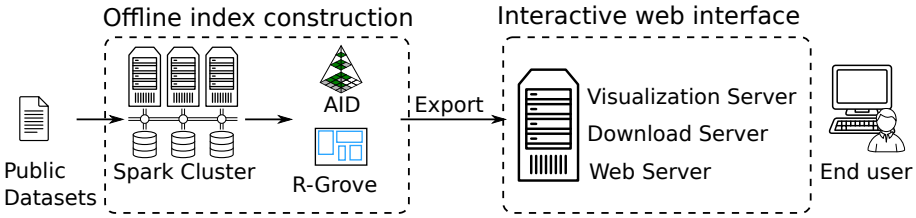


Figure 2: Overview of the system architecture of UCR-STAR

Figure 2 gives an architectural overview of UCR-STAR. The system consists of two major components, an offline index construction component and an interactive web-based interface. The offline index construction runs on a Spark cluster. It takes a publicly available dataset as input and produces two indexes, R-Grove and AID. The R-Grove index [12] is a spatial index for big spatial data that allows efficient retrieval of any rectangular query range. The AID [8] index is a light-weight adaptive visualization index that enables multilevel visualization of very large datasets. These two indexes are built on an in-house Spark cluster to scale to very large datasets. For example, a 100 GB dataset usually takes less than an hour to construct both indexes on a 12-node Spark cluster.

After the two indexes are constructed on Spark, they are then exported from the cluster into a single machine that hosts the second interactive web server. That single machine hosts three logical servers, a visualization server, a download server, and an Apache web server. The visualization server uses both indexes, AID and R-Grove, to serve *image tiles*; these are small images that are combined together to provide the map visualization. The download server provides the *subset download* features which allow users to download a part of the dataset in any format. The web server handles all requests to static files, e.g., HTML pages and images, and forwards tile and download requests to the corresponding servers. The next sections provide more details on how these two components work.

3 Offline Index Construction

The goal of the offline index construction process is to construct the R-Grove spatial index and the AID visualization index. Both are constructed on a Spark cluster to scale to big datasets.

3.1 Spatial Index Construction (R-Grove)

To be able to quickly access the data for very large datasets, we build a spatial index which is used by both the visualization and the download servers. R-Grove[12] is a novel partitioning and indexing method for big spatial data. R-Grove is an adaptation of the R-Tree family to big data. It inherits the quality index characteristics of R-Trees while balancing the load for distributed query processing. R-Grove is mainly a partitioning method that partitions the input dataset into 128 MB blocks. This is followed by constructing traditional local indexes for each partition. We use the RR*-tree [2] index for the local indexing step. The constructed index is stored as separate file, one for each partition, in the distributed file system accompanied with a small *master file* that stores the metadata of the partitions, e.g., the minimum bounding rectangle and the size of each one. Figure 3 shows an example of how the R-Grove global index partitions a 7.1 GB roads dataset into 60 partitions.

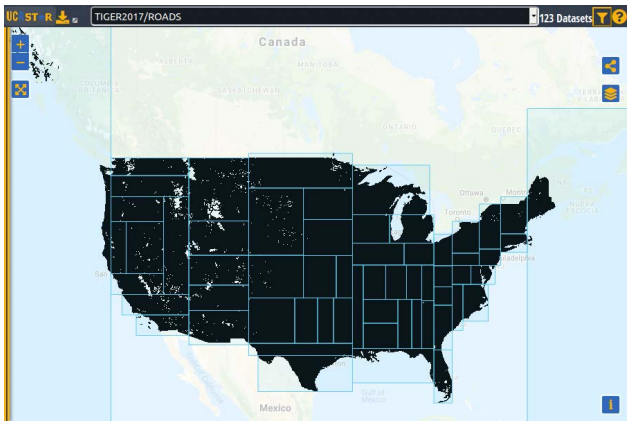


Figure 3: R-Grove index plot of a 7.1 GB roads dataset

3.2 Visualization Index Construction (AID)

To provide an interactive map visualization for the dataset, we use the standard tile-based pyramid structures illustrated in Figure 4. This structure is used by most web maps where the top tile covers the entire world and each deeper level multiplies the number of tiles by four. A standard pyramid of 20 levels will contain up to 366 billion tiles which is impractical to construct and maintain for hundreds of datasets. However, we make an observation that not all tiles are equal from the query processing perspective. For example, a tile representing the heavily populated New York City will have way more records than a tile representing the scarcely populated Palms Spring. In this case, it makes more sense to materialize the first tile as an image tile while the second one can be generated on demand.

In UCR-STAR, we use the Adaptive-Image-Data (AID) index [8] which builds on this observation to classify the tiles into four classes based on a threshold parameter (θ). The threshold θ represents the largest size that can be visualized on demand. Any tile that contains more θ records needs to be pre-generated and materialized to disk. Based on this threshold, AID defines the following four classes of tiles.

1. **Image tile** covers a large amount of data ($> \theta$) and is very expensive to visualize. This tile is pre-generated and materialized as an image.
2. **Data tile** has a small amount of data ($\leq \theta$) while its parent is an image tile. This tile can be retrieved and processed on-the-fly from the corresponding R-Grove index. If no spatial index is available, the records contained in this tile are stored in a *data file*, e.g., a CSV or GeoJSON file.
3. **Shallow tile** covers a small amount of data ($\leq \theta$) and its parent tile is not an image tile. This means that it is covered by an existing data tile and can also be generated on-demand.
4. **Empty tile** does not contain any records (zero records) and does not need to be stored. A prime example is a water area in a road network dataset.

The AID index that we use only contains the image tiles while all other tiles are generated upon request using the corresponding R-Grove index. The index construction process starts by building a histogram of the input data which is used to classify the tiles based on their sizes as explained above. After that, the image construction process uses the histogram to locate and generate only the image tiles.

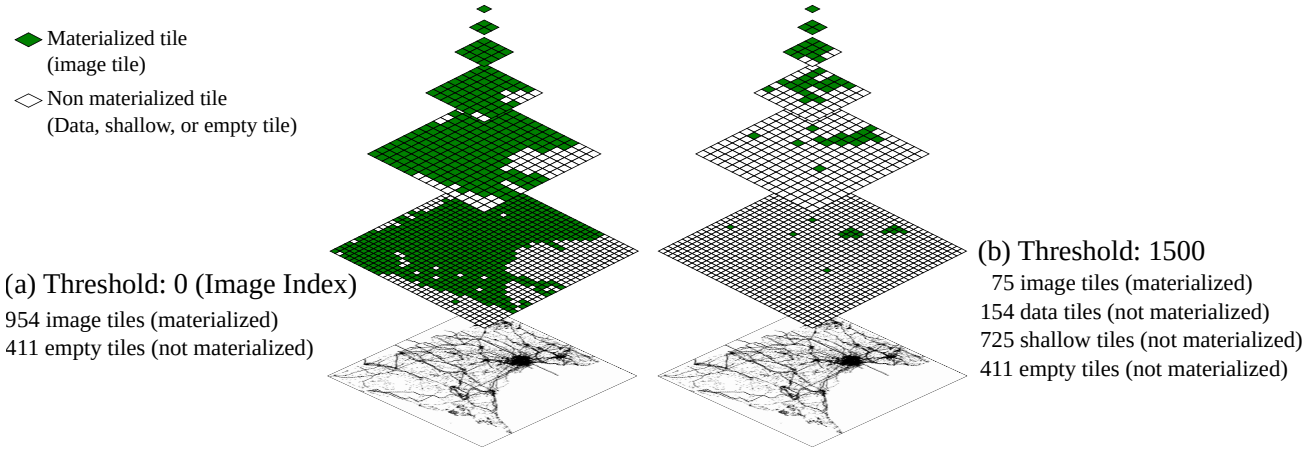


Figure 4: Examples of adaptive indexing with different threshold values

Figure 4 provides an example of the AID index, where (a) has a threshold 0, which means that all non-empty tiles are pregenerated and materialized as images, whereas in (b), where the threshold is 1500, only tiles having more than 1500 records are computed as image tiles while the rest are either data or shallow tiles and are not materialized. This not only reduces index size but also reduces the index construction time.

4 Interactive Web Interface

After the two indexes, R-Grove and AID, are constructed, they are exported from the distributed file system to a single machine that serves as the back-end server. This single physical machine hosts three server processes, visualization server, download server, and web server, described below.

4.1 Visualization Server

The visualization server serves a single query called GETTILE. This query requests a tile identified by a dataset ID, and a tile ID (z, x, y), where the dataset ID identifies which dataset is visualized and the tile ID represents the zoom level (z) and tile location in that level (x, y). The result is always a PNG image of size 256×256 pixels that the browser displays. As the user interacts with the maps, the browser sends a series of GETTILE queries to retrieve all the tiles that are visible on the screen and stitches them together to display the visible part of the map. The main challenge in this query is that not all the tiles are readily available as a PNG image on the server. Rather, some of them need to be generated on demand upon user request and they have to be served within 500 milliseconds to maintain the interactivity.

To answer the GETTILE query, the visualization server locates the AID index that corresponds to the dataset ID. First, it searches for a pre-generated PNG image by locating the file '`tile-z-x-y.png`'. If the file is found, then it is returned immediately. If no such file is available, then a tile needs to be generated on-the-fly. In this case, the corresponding R-Grove index is located. Then, through simple math, the minimum bounding rectangle (MBR) of the tile is calculated and a simple range query is executed on the R-Grove index. The resulting records are then visualized by simply scanning them. AID ensures that when a tile has to be generated on-demand, it is always small enough to be generated in less than 500 milliseconds.

To further improve the performance, the server caches all generated tiles in case they are requested again by other users. The server is configured to hold up to 100,000 tiles in the cache in PNG format which consumes an average of 100 MB of memory. Since its release, UCR-STAR served more than 820,000 tile requests with 27%, 48%, and 25%, static, on-demand, and cached tiles, respectively

4.2 Download Server

The download server empowers the users by allowing them to download a subset of the data based on an arbitrary rectangular range. While this feature is not new, UCR-STAR handles it in a unique way that was not done before. In short, it provides immediate download to any range of arbitrary size. To understand how this is different, we compare UCR-STAR to existing web sites that provide similar functionality.

- **GeoFabrick [7]** provides extracts from OpenStreetMap for prespecified ranges, e.g., countries or states. On the positive side, it provides immediate download to the files, but on the negative side it does not allow users to select arbitrary ranges and there is an overhead on the server on keeping all these files.
- **TAREEG [1]** follows a different approach that allows users to choose any arbitrary range and then it extracts the corresponding data and makes it available for download. However, it has a huge drawback in that it does not provide immediate access to the downloaded files. It asks the user to enter an email address and it sends the download link to that email after the file is ready. The extracted files are kept on the server for a month before they get deleted.
- **OpenStreetMap [10]** allows users to export an arbitrary rectangular range as an OSM file with immediate download. However, it only works for extremely small ranges that can be served on-the-fly. All requests for large regions, e.g., an entire city, are rejected.

UCR-STAR provides the best of all these options. It allows the user to choose any arbitrary range and the download starts immediately no matter how big the file is; even for hundreds of gigabytes of data. The trick is to to utilize the HTTP *chunking* feature which allows the response to be sent in chunks rather than in one piece. When the user request is received, the download server searches the R-Grove index to locate the partitions that match the query. As soon as the first partition is found, the server processes it and starts sending the data back to the client which makes the download start immediately. As the user downloads the data, the server keeps reading the next partitions and sends the data until the entire request is satisfied. This approach has many advantages over the traditional approach of serving static files including:

1. The download starts immediately regardless of the size.
2. The server does not need to store any additional files.
3. If the user cancels the download, the server immediately frees up the resources and does not continue the extraction.
4. The server can handle hundreds of concurrent download requests as each request is very light on resources.
5. The server can provide many file formats for download as the files are generated on-the-fly.

The only drawback is that the browser does not know the total size of the download. This means that there is no progress bar for the download. It also means that the download has to always restart from the beginning in case of a network failure. Since the release of this feature, UCR-STAR served 189 download requests with a total size of 87 GB. Note that the requests to download the full data are redirected to third party servers and are not tracked by UCR-STAR.

4.3 Web Server

As noted above, UCR-STAR uses two Java servers to host visualization and downloads. These servers could be running or hosted on one or multiple machines with arbitrary ports. On the other hand, an Apache Web server is available on the standard HTTP (80) and HTTPS (443) ports. Therefore, the Apache web server received all

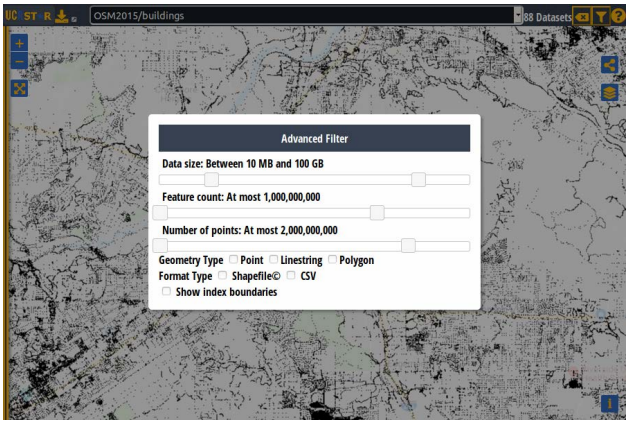


Figure 5: The search feature runs completely on the client side

visualization and download requests and it needs to forward them to the corresponding visualization or download server. In order to address this problem, we use the virtual host feature on Apache Web Server. In particular, based on the path after the domain, the request for visualizing a specific dataset is forwarded to the corresponding Java server. Similarly, a different path on the server is forwarded to the download server. This configuration allows one public domain to act as a single server for an array of services that can be hosted on single or multiple machines. This configuration also allows the web server to handle all static resources efficiently while the Java server can focus on the dynamic part of the application.

4.4 Front-end

This part briefly describes the set of HTML and JavaScript pages that are loaded into the end-user browser.

Map interface: We use OpenLayers [9] for displaying the map. It contains a base layer of either OpenStreetMap, Google Maps, or Google Satellite Imagery. On top of it, we put a tiled layer (XYZ Layer) that displays the dataset that is selected by the user. The XYZ layer is customized to send all the requests to the visualization server which hosts the tiles.

Search feature: The search functionality (Figure 5) is implemented entirely in JavaScript and is executed locally in the browser. Since the server currently hosts only a few hundred datasets, the browser can efficiently implement the search feature by scanning all these datasets for each query. This also relieves the server from addressing these additional search requests. A nice advantage to the users is that it adds some level of privacy as the server does not track the datasets that users search for.

Index display: One of the interesting features in UCR-STAR is the ability to show the underlying indexes on the datasets in a short amount of time regardless of the size of the dataset (Figure 3). To accomplish this, the front-end requests the *master file* of the selected index in GeoJSON format. The retrieved GeoJSON file is then used as a source for a vector layer that is added on top of the base and data layers. This layer is typically very small as it has one entry for each 128 MB partition. OpenLayers can efficiently handle a few thousand features in GeoJSON which is enough for all the datasets that are currently served by UCR-STAR. Since the client has the entire GeoJSON file, we allow the user to interact with the displayed index by clicking the partitions to show more details about them, e.g., total size and number of features in the partition.

Embed visualizations: Another interesting feature of UCR-STAR is the ability of users to *embed* a dataset visualization on another website without the need to install any additional software. Simply, UCR-STAR provides an HTML code snippet that users can copy/paste into their websites to get a fully interactive map display of a specific dataset similar to the one on the UCR-STAR website. This is a great option to share the datasets and provide richer engagement in the community. To implement the embed feature, UCR-STAR hosts a special JavaScript file that is designed to automatically initialize a map with the configured location and dataset visualization on top of it. The JavaScript file itself is static, but the HTML code snippet contains information about the geographical location, the zoom level to initialize the map, and the ID of the dataset to display on top of it. Once the map is loaded, OpenLayers handles all map interactions as usual.

Acknowledgments

This work is supported in part by the National Science Foundation (NSF) under grants IIS-1838222 and CNS-1924694 and by the USDA National Institute of Food and Agriculture, AFRI award number A1521.

References

[1] L. Alarabi, A. Eldawy, R. Alghamdi, and M. F. Mokbel. TAREEG: a mapreduce-based system for extracting spatial data from openstreetmap. In *Proceedings of the ACM Symposium on Advances in Geographic Information Systems, ACM SIGSPATIAL*, pages 83–92, Dallas/Fort Worth, TX, Nov. 2014.

[2] N. Beckmann and B. Seeger. A Revised R*-tree in Comparison with Related Index Structures. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, pages 799–812, Providence, RI, July 2009.

[3] Big Ten Academic Alliance Geoportal. `https://geo.btaa.org/`.

[4] Data.gov. The home of the U.S. Governments open data. `https://www.data.gov/`.

[5] Data.gov.uk. `https://www.data.gov.uk/`.

[6] Los Angeles - Open Data Portal, 2019. `https://data.lacity.org/`.

[7] GeoFabrik Homepage, 2019. `https://www.geofabrik.de/`.

[8] S. Ghosh, A. Eldawy, and S. Jais. AID: An Adaptive Image Data Index for Interactive Multilevel Visualization. In *Proceedings of the IEEE International Conference on Data Engineering, IEEE ICDE*, Macau, China, Apr. 2019.

[9] OpenLayers API: A high-performance, feature-packed library for all your mapping needs, 2019. `https://openlayers.org/`.

[10] OpenStreetMap, 2019. `https://www.openstreetmap.org/`.

[11] United Nations Open Data - A World of Information, 2019. `http://data.un.org/`.

[12] T. Vu and A. Eldawy. R-Grove: Growing a Family of R-trees in the Big-data Forest. In *Proceedings of the ACM Symposium on Advances in Geographic Information Systems, ACM SIGSPATIAL*, pages 532–535, Seattle, WA, Nov. 2018.

[13] Yahoo! Webscope Datasets, 2018. `https://webscope.sandbox.yahoo.com/`.