Assignment 2: Report

Sajja Patel - 2021101107, Bhumika Joshi - 2022121006

August 2024

1 Data Preprocessing

In order to do any analysis, we first need to understand the the data that we are working with. We also need to make sure that the data is available in a usable format, doesn't have inconsistencies, null values, or outliers that affect our analysis. We also need to identify useful features from the redundant and non-essential features in our dataset. All this can be done during the data preprocessing step.

2 Attribute-Oriented Induction

Attribute-Oriented Induction (AOI) in data mining is a data generalization method used to simplify large datasets by summarizing them at higher levels of abstraction. It involves:

- Generalizing attributes: Specific attribute values are replaced with more general concepts.
- Reduction of data: By generalizing, the number of records in a dataset is reduced, making patterns easier to discover.

AOI is mainly used in concept description, summarization, and decision tree construction. Here, we have taken the following steps to implement AOI:

- **Attribute Removal**: Attributes that are too specific and unique for each entry are removes because they cannot provide adequate information about patterns. The attributes DOL Vehicle ID and VIN (1-10) were removed.
- **Attribute Generalization**: This was carried out as:
  - In the case where concept hierarchy exists, generalize to the root of the hierarchy.
  - In cases like that of Postal Codes, grouping can be done by considering only the initial (significant) digits.
  - In other cases, the generalization can be done by grouping the values into bins or ranges.
- **Extracting Characteristic Rules**: By forming crosstabs of the attributes that we have obtained so far, we can extract rules in the form of if-then statements that help us extract likely patterns in the data.

3 BUC Algorithm Implementation

The BUC (Bottom-Up Clustering) algorithm is used for efficiently computing iceberg cubes in OLAP data structures. It starts from the most specific data points (bottom) and aggregates them upwards to higher levels of granularity. The algorithm focuses on computing only those data aggregates that meet a predefined threshold (iceberg condition), making it efficient for handling large datasets by avoiding unnecessary computations on less significant data. It is commonly applied in multidimensional data analysis for tasks like summarization and clustering.

3.1 In-Memory Implementation

3.1.1 Code Explanation

1. Binning: The first part of the code focuses on *binning* two columns from the dataset: `Base MSRP` and `Electric Range`.

- The `Base MSRP` column is binned into the following categories:0, (1-50k), (50-100k), (100-200k), (200k+).
- The `Electric Range` column is similarly binned into: 0, (1,50), (50,100), (100-200), (200+).

This step helps in categorizing the data into meaningful ranges for further analysis.
2. Aggregation Function: The `aggregate` function computes the *unique values* from the first column of data and returns the unique attributes along with their *counts*. This is essential for determining the frequency of different attribute values during the BUC computation.

3. BUC Algorithm: The core of the code implements the *Bottom-Up Clustering (BUC) algorithm*:

- The `compute_cube` function recursively computes the data cube for each subset of dimensions.
- It processes the data by:
  - Sorting it based on the first column,
  - Aggregating unique values for the first column,
  - Recursively breaking down the dataset along the remaining dimensions.
- Additionally, it handles the "ALL" case, which represents the inclusion of all possible values for a given dimension.

4. Minimum Support: The algorithm is run for *different minimum support (min_sup) values* to control the *granularity* of the clustering process:

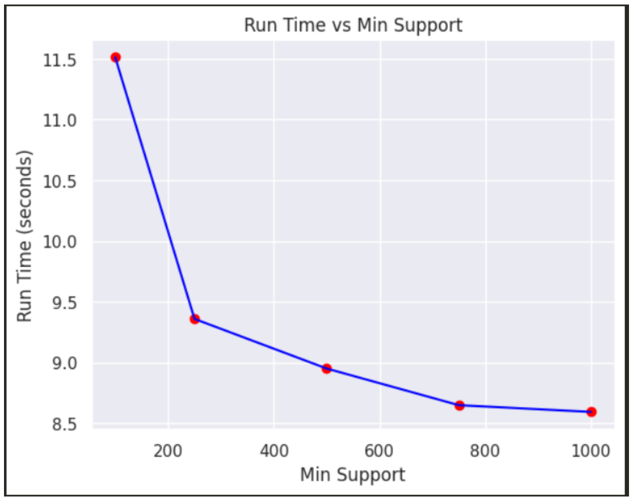
- Only subsets with at least `min_sup` instances are considered significant, and results are pruned accordingly.
- The BUC algorithm is run for each `min_sup` value, with execution time and memory usage tracked.

5. Performance Tracking: For each `min_sup` value, *Execution time* and *peak memory usage* are recorded. Results for `min_sup = 100` are saved to a CSV file (`result.csv`).

3.1.2 Output and Observations

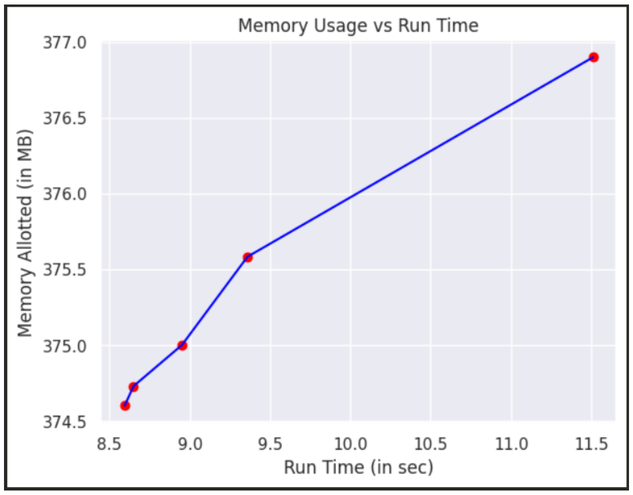
Runtime vs min_sup

- The runtime is slightly longer for lower `min_sup` values and tends to decrease as `min_sup` increases.
- **Reason:** Lower `min_sup` values result in a larger number of sub-cubes being considered, leading to more computation. As `min_sup` increases, fewer sub-cubes meet the support threshold, reducing the computation time.



Memory usage vs min_sup

- The memory usage is highest for the lowest min_sup value (100) and decreases as min_sup increases.
- **Reason**: Similar to runtime, a lower min_sup value means that more sub-cubes are generated and stored, requiring more memory. As min_sup increases, fewer sub-cubes are kept in memory, leading to reduced memory usage.



3.2 Out-of-Memory Implementation

3.2.1 Code Explanation

In this section, I evaluated the impact of available memory size on the runtime of the Bottom-Up Clustering (BUC) algorithm. The following steps outline the process I followed:

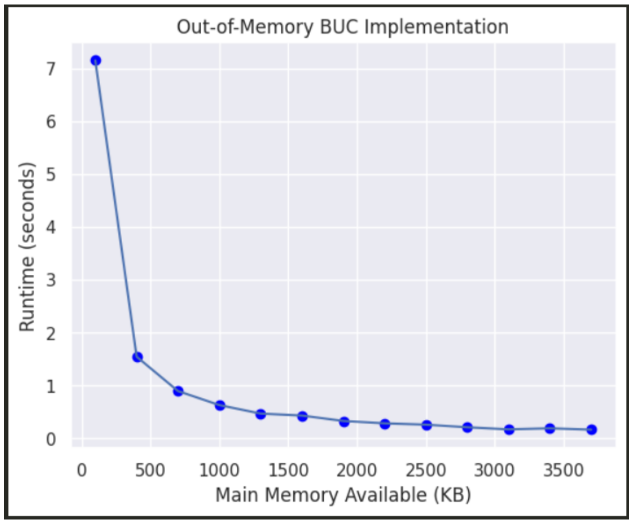
- **Memory Usage Calculation:** I calculated the total memory usage of the dataframe (`df`) in kilobytes (KB) to determine the size of the data in memory.
- **Paging Simulation Setup:**
  - determined the number of columns in the dataframe to configure how data would be sliced.
  - initialized lists to track memory sizes and corresponding runtime measurements.

- **Paging Simulation:**
  - Simulated paging by varying the memory sizes from 100 KB to 4000 KB in increments of 300 KB.
  - For each memory size, calculated the number of data pages that could be processed.
  - Processed each page of data, applying the BUC algorithm to simulate handling chunks of data within the available memory.
- **Performance Measurement:** I measured the runtime required to process data with each memory size configuration and recorded the runtime and memory size for each iteration.

This approach helped me assess how varying the amount of memory available influences the efficiency of out-of-memory data processing, particularly for large datasets.

3.2.2 Output and Observations

- The output aligns with what we expect from an out-of-memory implementation.
- Memory Sizes (KB): we have a range of memory sizes increasing from 100 KB to 3700 KB in steps of 300 KB. This is in line with the paging simulation where more memory allows larger "pages" of data to be processed at once.
- Run Times (secs): the times generally decrease as the memory size increases, which makes sense because larger memory means fewer pages, leading to less overhead and faster execution.
- There are slight fluctuations (e.g., at 2800 KB vs 2500 KB), but its possible in real-world scenarios due to system and memory management factors.



4 Optimization Technique

For optimizing the process of data cubing, we have implemented an Iceberg Cube with the following optimizations:

- **Sorting Dimensions by Cardinality** Sorting the dimensions by cardinality (i.e., the number of unique values in each dimension) before processing can drastically reduce the number of unnecessary computations. By processing dimensions with fewer unique values first, we can prune the search space more effectively, reducing the overall number of cuboids that need to be computed.

- **Caching Intermediate Results** Caching intermediate results allows the algorithm to reuse previously computed aggregates instead of recalculating them. Since the BUC algorithm processes multiple cuboids that may share partial results, storing these intermediate computations avoids redundant operations.
- **Apriori Pruning** The Apriori principle allows us to prune cuboids based on their parent cuboid. If a cuboid does not meet the minimum support (i.e., the count of occurrences is below a certain threshold), then its child cuboids (more granular aggregations) will also not meet the minimum support. This allows for early termination of certain branches in the cube computation, saving significant time.

After implementing the optimized iceburg cube, we compare it with the non-optimized version and compare the performance. The performance of both the algorithms in the given benchmarks are as follows:

- Non-Optimized BUC
  - **Execution time**: 593.6238 seconds
  - **Memory usage**: 4354.17 MB
  - **No. of Cuboids**: 4066640
- Optimized Iceberg BUC
  - **Execution time**: 0.6824 seconds
  - **Memory usage**: 503.21 MB
  - **No. of Cuboids**: 996

As we can see, computation time is drastically reduced when using optimized iceberg BUC as compared to the non-optimized BUC cube technique. There is also significant reduction in the memory usage in the optimized version. Also, the no. of cuboids computed in the optimized version is far smaller than in the non-optimized version.

5 Comparison of BUC and AOI

5.1 Primary Purposes and Use Cases

5.1.1 BUC

Purpose: BUC is designed for efficient computation of a data cube for OLAP (Online Analytical Processing). It aims to aggregate data across multiple dimensions and is used in data warehousing and multidimensional databases to support decision-making and data analysis.
Use Case: BUC is primarily used when you need to perform multidimensional aggregation over a large dataset. For example, computing the count of cars per State, County, Make, Model and City.

5.1.2 AOI

Purpose: AOI is a data generalization technique used in data mining to abstract data by replacing specific attribute values with more general ones. It is used to find higher-level patterns or concepts in large datasets.
Use Case: AOI is typically used in concept description tasks such as summarizing data in more general terms, often for decision tree learning, clustering, or summarizing large sets of transactions in a more comprehensible way. For, example Summarizing Electric Ranges of Vehicles in groups like '0-50', '51-100', '101-150', '151-200', etc. that can be interpreted as very low, low, medium, high, etc.

5.2 Types of Insights or Patterns

5.2.1 BUC

Insights: BUC is best suited for discovering aggregated patterns across multiple dimensions, like trends, correlations, and outliers. It computes data cubes and can highlight regions in a dataset that meet certain criteria (e.g., sales exceeding a threshold).
Example: For example, cars in the county King (94460) vastly outnumber those in the county Skamania (196).

5.2.2 AOI

Insights: AOI is effective for discovering generalized patterns, typically by summarizing data at a higher abstraction level. It is used to find concept hierarchies and can extract more meaningful patterns from raw data.
Example: AOI could show that the '0-50' electric range has the highest instances of the lowest Base MSRP bracket of '0-30,000' (128956 instances)

5.3 Computational Efficiency and Scalability

5.3.1 BUC

Efficiency: BUC can be computationally expensive, especially with high-dimensional data, because it explores and aggregates across all possible cuboids. However, it employs pruning techniques like Apriori to improve efficiency by focusing only on the cuboids that meet certain criteria (like minimum support).
Scalability: BUC scales reasonably well with optimizations like sorting dimensions by cardinality and caching intermediate results, but its complexity grows with the number of dimensions and the size of the data.
Example: In the cars dataset, BUC calculates the count of cars in a particular County, having a particular Make, etc. but only in cases which meet the minimum support threshold criteria (County King meets the min. sup. criteria, while County Skamania does not meet this criteria with any possible Make. It is fast if we use limited dimensions but cube computation time increases drastically as we consider more and more dimensions.

5.3.2 AOI

Efficiency: AOI is generally more efficient than BUC because it performs data reduction by summarizing or generalizing attributes. It doesn't compute aggregates for all possible combinations but instead generalizes data into broader categories.
Scalability: AOI is highly scalable for large datasets because it works by reducing the data's dimensionality through abstraction rather than aggregating data across multiple dimensions.
Example: For a dataset with detailed car attributes, AOI would drop VIN attribute because it is too unique and does not provide meaningful information and also group attributes like Make, Electric Range, Base MSRP, etc.

5.4 Interpretability of the Results

5.4.1 BUC

Interpretability: The results from BUC are highly interpretable for users familiar with OLAP-style reports. The output is a data cube, where users can easily drill down or roll up to explore different levels of aggregation. While the results are structured, the complexity grows with the number of dimensions, making it harder to interpret all combinations of aggregated data if many dimensions are involved.
Example: A user can interpret the total count for "County King" and "Make TESLA" in "City Seattle" but as more dimensions are added, the cube becomes harder to navigate.

5.4.2 AOI

Interpretability: AOI produces highly interpretable results by summarizing and generalizing the data into more human-readable categories. The generalization helps users grasp high-level patterns without needing to drill into specific combinations. Since AOI reduces dimensionality by generalizing data, the results are easier to interpret but may lose granularity.
Example: A user can easily interpret that "Low Electric Range" correlates with "Low Base MSRP" , but may not see the exact breakdown of this pattern when considered along with other attributes like Make and Year.

5.5 Scenarios Where One Might Be Preferable Over the Other

5.5.1 BUC

Scenario: When you need detailed multidimensional aggregation and exploration of data across various levels of granularity. If the goal is to support decision-making through OLAP cubes where users can drill into or roll up the data, BUC is ideal.
Example: For someone planning sales of cars, the amount of cars of different types present in different location may help them understand popularity of cars and help them determine the best sales strategies.

5.5.2 AOI

Scenario: When the goal is to summarize large amounts of detailed data into more abstract, human-readable insights. AOI is useful for generating reports that highlight broader patterns without needing the user to manually drill into specifics.
Example: In customer segmentation, AOI can help generalize a dataset by grouping cars based on Model Year, Electric Range, etc, and see how these correlate with Clean Fuel Eligibility, Base MSRP values, etc.