UNIVERSITÀ DEL SALENTO

Facoltà di Ingegneria

Degree Course in Computer Engineering

Project Documentation

Data Mining and Machine Learning

Implementation of Fast and accurate mining of correlated heavy hitters

Professor

Prof. Massimo Cafaro

Student

Farah Murtaza

Index

Chapter 1 – Introduction

In the era of big data, real-time analytics and efficient processing of streaming data have become paramount for various applications, ranging from network monitoring to financial transactions and beyond. The Cascading Space Saving Correlated Heavy Hitters (CSSCHH) algorithm emerges as a powerful solution to address the challenges posed by the identification of correlated heavy hitters in dynamic data streams.

Background and Motivation:

The CSSCHH algorithm builds upon the foundational concepts of the Space Saving algorithm introduced by Metwally et al. in 2006. The Space Saving algorithm is renowned for its ability to maintain accurate frequency estimates of items in a data stream with limited memory usage. CSSCHH extends this approach to tackle the intricate task of identifying correlated heavy hitters – pairs of items whose frequencies exhibit significant correlations.

Key Objectives:

The primary objective of CSSCHH is to efficiently and accurately identify correlated heavy hitters in real-time streaming data. Correlated heavy hitters, in this context, refer to pairs of items whose co-occurrence frequencies surpass certain predefined thresholds. The algorithm employs two distinct Space Saving stream summaries, one for primary items and another for correlated tuples, to achieve this goal.

Methodology:

CSSCHH operates by maintaining two independent Space Saving summaries, denoted as Sp and St. The Sp summary focuses on monitoring a subset of primary items, while the St summary tracks a subset of tuples that appear in the data stream. The algorithm employs a lightweight data structure to keep elements sorted by their estimated frequencies, enabling efficient identification of heavy hitters.

Significance of the Algorithm:

The significance of CSSCHH lies in its capability to provide accurate and real-time insights into the correlations between items in a streaming environment. This has broad implications across diverse domains, including network traffic analysis, where the identification of correlated events is crucial for anomaly detection, and in financial systems, where understanding correlated transactions is essential for fraud detection.

Overview of the Documentation:

This documentation serves as a comprehensive guide to understanding the principles, implementation, and performance characteristics of the CSSCHH algorithm. It covers the initialization, update, and querying procedures, supported by theoretical correctness theorems. Additionally, the document delves into the algorithm's space and time complexity, offering insights into its efficiency in handling large-scale streaming data

The subsequent sections of this documentation will provide an in-depth exploration of CSSCHH, including its algorithmic details, theoretical underpinnings, and practical considerations for implementation and usage.

Chapter 2 - Algorithm Overview

The Space Saving algorithm is a fundamental concept in the CSSCHH algorithm. In brief, a stream summary **S** is a data structure used to monitor **N** distinct items and includes **k** counters. These counters are designed to monitor k distinct items, updating their frequencies as the items appear in the streaming data. The update procedure of the Space Saving algorithm is a key element, ensuring the accurate tracking of item frequencies in real-time. When processing an item, the algorithm checks if it is already monitored by a counter. If so, the estimated frequency of the item is incremented. If the item is not currently monitored, and a counter is available, it becomes responsible for monitoring the item with an initial frequency of one. In case all counters are occupied, the counter storing the item with the minimum frequency is incremented by one, and the monitored item is replaced with the new one. The sum of the frequencies of all monitored items in S is equal to the length of the input stream, denoted as N. Furthermore, the Space Saving algorithm

ensures that the error in estimating the frequency of an item is bounded by the minimum frequency in S. The Space Saving algorithm's efficiency and accuracy in frequency estimation lay the groundwork for its integration into the CSSCHH algorithm, enabling the identification of correlated heavy hitters in streaming data.

Chapter 3 - Initialization phase

The initialization phase of the CSSCHH algorithm is a crucial step where the primary and correlated stream summaries, denoted as *Sp* and *St*, are established and configured. This phase sets the foundation for subsequent processing and ensures that the algorithm is appropriately equipped to identify correlated heavy hitters in the streaming data.

Explanation:

1. **Input Parameters:**
    - $\phi 1$: Threshold for primary items.
    - $\phi 2$: Threshold for correlated items.
    - $\varepsilon 1$: Tolerance for primary items.
    - $\varepsilon 2$: Tolerance for correlated items.
2. **Constants β and γ:**

    Calculate β and γ based on the formula
3. **Counter Allocation:**
    - $k1$ counters are allocated for the primary stream summary *Sp*.
    - $k2$ counters are allocated for the correlated stream summary *St*.
4. **Return:**
    - The initialized primary (*Sp*) and correlated (*St*) stream summaries are returned.

The calculation of $k1$ and $k2$ involves determining the appropriate sizes for the primary and correlated stream summaries based on the input thresholds and tolerances. The objective is to strike a balance between accuracy in detecting primary frequent items and efficiency in memory usage.

The values of $k1$ and $k2$ are carefully chosen to meet constraints related to false positives and the number of real frequent items. These constraints are expressed through equations involving thresholds, tolerances, and constants like β and γ.

```
void SpaceSaving::initialize() {
    beta = 1 / (epsilon2 * phi1);
    gamma = (epsilon2 + phi2) / (epsilon2 * phi1);

    k1 = static_cast<int>(std::max(1 / epsilon1, gamma + std::sqrt(beta * gamma)));
    k2 = static_cast<int>(beta * k1 / (k1 - gamma));

    counterMapSp.clear();
    counterMapSt.clear();
    counterMapSp.reserve(static_cast<std::size_t>(k1));
    counterMapSt.reserve(static_cast<std::size_t>(k2));
}
```

Chapter 4 – Update Procedure

The Update Procedure in the CSSCHH algorithm, as outlined in Algorithm 2, plays a crucial role in maintaining the accuracy of the primary (Sp) and correlated (St) stream summaries as new items and tuples appear in the streaming data. This procedure ensures that the counters are appropriately updated to reflect the evolving frequencies of items and tuples.

```
Require: x, y, the items of a tuple.
Ensure: Update of S_p and S_t stream summaries.

1: procedure CSSCHH-Update(S_p, S_t, x, y)
2: SpaceSavingUpdate(S_p, x)
3: SpaceSavingUpdate(S_t, (x, y))
4: end procedure
```

Explanation:

1. **Input Parameters:**
    - *x*: The primary item.
    - *y*: The correlated item in a tuple (*x*,*y*).
2. **SpaceSavingUpdate for *Sp*:**
    - The primary item *x* is used to update the primary stream summary *Sp*.
    - The SpaceSavingUpdate procedure is invoked for *Sp* with the primary item *x*.
    - This ensures that the counter for the primary item *x* in *Sp* is appropriately updated.
3. **SpaceSavingUpdate for *St*:**
    - The tuple (*x*,*y*) is considered as a single item and is used to update the correlated stream summary *St*.
    - The SpaceSavingUpdate procedure is invoked for *St* with the tuple (*x*,*y*).
    - This ensures that the counter for the tuple (*x*,*y*) in *St* is appropriately updated.
4. **End of Procedure:**
    - The Update Procedure concludes after both *Sp* and *St* have been updated based on the appearance of the primary item *x* and the tuple (*x*,*y*) in the streaming data.

The actual updating of counters for both Sp and St is performed by the SpaceSavingUpdate procedure, which is a fundamental part of the CSSCHH algorithm. The SpaceSavingUpdate procedure is responsible for maintaining the stream summaries by incrementing counters for existing items or adding new items when necessary, all while adhering to the principles of the Space Saving algorithm. This Update Procedure ensures that the CSSCHH algorithm remains adaptive to changing frequencies in the streaming data, contributing to the accurate identification of correlated heavy hitters.

```
void SpaceSaving::update(const std::pair<char, char>& tuple) {
    SpaceSavingUpdate(counterMapSp, tuple.first);
    SpaceSavingUpdate(counterMapSt, tuple);
}
```

```cpp
void SpaceSaving::SpaceSavingUpdate(std::unordered_map<char, int>& counterMap, const char& item) {
    auto it = counterMap.find(item);
    if (it != counterMap.end()) {
        it->second++;
    } else {
        if (counterMap.size() < k1) {
            counterMap.emplace(item, 1);
        } else {
            auto minCountItem = std::min_element(counterMap.begin(), counterMap.end(),
                                                 [](const auto& lhs, const auto& rhs) {
                                                     return lhs.second < rhs.second;
                                                 });
            counterMap.erase(minCountItem);
            counterMap.emplace(item, 1);
        }
    }
}
```

Chapter 5 – Querying Correlated Heavy Hitters

The process of querying correlated heavy hitters (CHHs) is a crucial aspect of the Cascading Space Saving algorithm (CSSCHH). The algorithm employs a querying mechanism to identify and report primary items and correlated tuples whose frequencies exceed specified thresholds.

Querying Process Explanation:

- **Primary Item Check (*Sp*):**
  - The algorithm first scans through the counters in the primary stream summary *Sp*.
  - For each counter c_p_j, representing a primary item *r*, it checks if the estimated frequency $\hat{}fr$ exceeds the threshold.
  - If $\hat{}fr$ is greater than threshold, the primary item *r* and its estimated frequency $\hat{}fr$ are added to the list *F*.
- **Correlated Tuple Check (*St*):**
  - Next, the algorithm scans through the counters in the correlated stream summary *St*
  - For each counter c_t_j, representing a correlated tuple (*r*,*s*), it checks if the primary item *r* is present in the list *F* and if the specific condition holds.
  - If both conditions are met, the correlated tuple (*r*,*s*,$\hat{}frs$) is added to the set *C*.
- **Result:**
  - The set *C* contains the identified correlated frequent items and is returned as the result of the query.

The querying process ensures that only those primary items and correlated tuples exceeding their respective frequency thresholds are reported, contributing to the accurate identification of correlated heavy hitters in the streaming data.

```
void SpaceSaving::querySt() {
    std::unordered_set<char> primaryItems;

    // Calculate the total frequency in the primary stream summary (N)
    int totalFrequencySp = 0;
    for (const auto& entry : counterMapSp) {
        totalFrequencySp += entry.second;
    }

    // Step 1: Collect primary items and their frequencies from Sp
    for (const auto& entry : counterMapSp) {
        if (entry.second > phi1 * totalFrequencySp) {
            primaryItems.insert(entry.first);
        }
    }

    // Step 2: Check for correlated items in St
    for (const auto& entry : counterMapSt) {
        char r = entry.first.first;
        char s = entry.first.second;
        int frs = entry.second;

        // Check if r is a primary frequent item and if frs > phi2 * (fr - N/k1)
        if (primaryItems.count(r) && frs > phi2 * (entry.second - totalFrequencySp / k1)) {
            // Correlated heavy hitter detected
            std::cout << "Correlated heavy hitter: (" << r << ", " << s << ", " << frs << ")" << std::endl;
        }
    }
}
```

Chapter 6 – Space and Time Complexity

In this section, we delve into the analysis of the space and time complexity of the Cascading Space Saving algorithm (CSSCHH). Understanding the computational efficiency of the algorithm is crucial for assessing its performance in real-time streaming scenarios.

Time Complexity

Initialization Phase (Algorithm 1):

The initialization phase involves allocating counters for the primary stream summary *Sp* and the correlated stream summary *St*. This phase's complexity is $O(1)$ since it consists of a few assignments and computations

Update Procedure (Algorithm 2):

The update procedure is executed for each item or tuple in the stream. The key operations involve updating counters in *Sp* and *St*, with each operation having a worst-case complexity of $O(1)$. Therefore, the overall time complexity of the update procedure is $O(1)$.

Querying Correlated Heavy Hitters (Algorithm 3):

The querying process involves scanning through counters in both *Sp* and *St*. The worst-case scenario is linear, as it requires inspecting each of the $k1$ counters in *Sp* and each of the $k2$ counters in *St*. Therefore, the time complexity of the querying process is $O(k1+k2)$.

Space Complexity

The space complexity of an algorithm is primarily determined by the memory requirements for maintaining data structures. In the case of CSSCHH:

- **Primary Stream Summary (*Sp*):** Requires $k1$ counters.
- **Correlated Stream Summary (*St*):** Requires $k2$ counters.

Therefore, the total space complexity is $O(k1+k2)$.

Summary:

- **Initialization Phase:** $O(1)$
- **Update Procedure:** $O(1)$
- **Querying Process:** $O(k1+k2)$
- **Space Complexity:** $O(k1+k2)$

The CSSCHH algorithm's time and space complexities are advantageous for real-time streaming applications, as they remain constant or linear with respect to the number of counters used in the stream summaries. This efficiency is crucial for processing streaming data efficiently and identifying correlated heavy hitters in a timely manner.

Chapter 7 – Conclusions

The Cascading Space Saving algorithm (CSSCHH) stands out as a robust solution for real-time data stream analysis. Rooted in the Space Saving algorithm, CSSCHH offers a sophisticated approach to monitoring frequent items and correlated pairs within dynamic data streams.

CSSCHH's strength lies in its ability to efficiently track individual items and pairs concurrently. By leveraging Space Saving techniques, it effectively identifies popular items and discerns patterns of co-occurrence. The algorithm employs intelligent counter management, dynamically adjusting resource allocation based on specific criteria. This ensures a balance between accuracy and resource efficiency.

One notable feature of CSSCHH is its adeptness at swiftly searching through stored information. When seeking significant connections between items, CSSCHH efficiently scans its data to deliver valuable insights. Moreover, the algorithm provides theoretical assurances, promising minimal instances of missing important patterns (false negatives) and controlling the occurrence of unimportant results (false positives). CSSCHH is resource-conscious, designed to operate efficiently without excessive memory usage or prolonged processing times. Its suitability for applications requiring prompt insights from streaming data positions it as a valuable tool in the era of escalating real-time data demands. In essence, CSSCHH functions as an intelligent detective for streaming data, offering a quick and reliable means to extract meaningful information from the deluge of data in contemporary technological landscapes.