A System for In-Network Hot Motion Path Detection over RFID and Sensor Network Technologies

Christodoulos Kampanis

March 2025

Abstract

In today's interconnected world, vast amounts of data traverse networks along diverse paths, generating an immense flow of information. In this fast-paced information age, where speed is paramount and delays can be detrimental, the demand for efficient data processing has reached unprecedented levels. Achieving efficiency requires balancing resource utilization with accuracy and processing speed. This necessity drives the pursuit of data compression techniques that preserve critical information while reducing transmission overhead. One of the most demanding applications of this principle is in road networks, where roads form the network structure, and moving objects represent the continuous flow of information.

This work introduces a distributed algorithm based on *MinHashing* to identify frequent road network edges and hot motion paths, reducing communication costs while preserving data accuracy. By leveraging RFID-enabled motes, the algorithm efficiently compresses data for real-time traffic analysis, highlighting trade-offs between detection accuracy and communication efficiency. These findings contribute to scalable, decentralized traffic monitoring systems, addressing challenges in data compression, privacy, and transmission efficiency.

Acknowledgements

I would like to express my earnest thanks to my supervisor, Dr. Nikos Giatrakos, who has provided tremendous support and guidance throughout my work. His invaluable insights, constructive feedback, encouragement and expertise have inspired me and i am profoundly grateful for his mentorship. My gratitude also extends to my parents for their unconditional support and encouragement. Their belief in me, invaluable suggestions, and full support in all my decisions have been foundational. No words could possibly express my deepest gratitude for their endless love, self-sacrifice, and unwavering help. To them and my partner, I dedicate my thesis. Finally, I would like to express my heartfelt appreciation to the Technical University of Crete (TUC) for providing the academic environment, resources, and opportunities that made this work possible.

Contents

Chapter 1

Introduction

The integration of Wireless Sensor Networks (WSNs) and Radio Frequency Identification (RFID) technologies has gained significant attention for addressing monitoring and surveillance challenges in various domains [1, 2, 3]. While WSNs are widely recognized for their distributed processing capabilities, RFID technology excels in object identification and tracking, making their combination highly beneficial for real-time applications such as traffic monitoring.

Traditional systems for traffic monitoring, such as those based on GPS, face notable limitations. These include high costs associated with continuous mobile network dependency, privacy concerns due to constant location tracking, and scalability challenges in urban environments. Conversely, RFID-enabled systems offer a cost-effective and scalable alternative. For example, RFID transponders deployed in the San Francisco Bay Area successfully provided real-time traffic data by equipping vehicles with passive RFID tags [4]. Similarly, over 10,000 sensor nodes powered by batteries or solar panels were deployed across California roadways to monitor traffic flows continuously [13].

The deployment of RFID-enabled motes along road networks enables the collection of detailed traffic data by interacting with RFID tags on vehicles. Unlike GPS-based systems that often provide aggregate data, this approach identifies specific "hot motion paths"—frequently traversed routes that are critical for applications such as dynamic traffic management, accident prevention, and emergency response [14]. The focus of this thesis is on the technical challenges associated with managing and processing the large volumes of data generated by these systems, particularly concerning data transmission and compression.

Compression is essential to reduce the bandwidth and storage requirements while preserving data accuracy. Traditional lossy compression methods are unsuitable due to the risk of losing critical identifiers, which are necessary for reliable traffic analysis. Instead, this thesis employs *MinHashing*, an efficient technique for data transformation that ensures privacy, reduces redundancy, and supports scalable analysis [20, 21].

MinHashing offers several advantages:

- **Data Privacy:** It prevents the reconstruction of vehicle identifiers, addressing privacy concerns.
- **Compression Ratio:** It reduces data size by generating compact representations, suitable for large-scale systems.
- **Efficiency:** MinHashing's low computational overhead enables real-time processing.
- **Redundancy Handling:** It minimizes redundancy by producing unique signatures for datasets.
- **Comparability of Data:** MinHash signatures retain the ability to compare datasets, enabling the detection of "hot motion paths" using similarity metrics such as the Jaccard index.

The detection of high-traffic routes involves identifying "hot motion paths" as supersets of smaller motion edges. This process utilizes the principles of transitive closure to extract meaningful patterns from distributed data. By addressing challenges in transmission efficiency and data compression, this research contributes to the development of scalable and decentralized traffic monitoring systems.

Our distributed *MinHashing*-based algorithm compresses the transmitted data, consisting of vehicle IDs, and its performance is measured in terms of accuracy and communication costs. To ensure an unbiased assessment, we deliberately avoid advanced collision-avoidance techniques such as Frequency Division Multiplexing (FDM) or Time Division Multiplexing (TDM) [24, 23]. Instead, we compare the extent and impact of data collisions in raw data with the outcomes observed after applying our algorithm.

By examining collisions in both scenarios, we gain insights into the trade-offs between accuracy and efficiency, evaluating the algorithm's ability to reduce communication overhead while preserving data quality under real-world constraints. This approach highlights its robustness and practical utility for decentralized systems.

Chapter 2

Related Work

Previous efforts integrating RFID and WSN technologies have demonstrated their potential in large-scale monitoring and data analysis. Zhang et al. [1] and Liu et al. [3] explored architectures and challenges in combining these technologies. For instance, RFID readers embedded on mote devices enable scalable tracking and monitoring without extensive infrastructure.

Flowscan [4] identifies hot routes through offline traffic density analysis by clustering road segments with shared vehicle trajectories. While effective for post-hoc pattern discovery, it relies heavily on centralized processing, limiting responsiveness to changing traffic conditions. In contrast, our system leverages WSN and RFID technologies to perform distributed, real-time detection of hot motion paths. Additionally, where Flowscan avoids data collisions by aggregating data offline, our method embraces and analyzes collisions , and utilizes them to analyze the interplay between road edge popularity and data compression, providing live, low-cost insights into traffic patterns.

LiveMap [5] enhances vehicular perception by enabling real-time object detection and sharing between connected vehicles. This focuses on localized, vehicle-level awareness differs from our system's goal of analyzing broader traffic patterns across road networks. While LiveMap improves navigation and driver safety, it lacks the capability to monitor large-scale traffic flows or identify congested paths, which our approach achieves through aggregate motion path analysis.

SPIRE [6] applies compression and inference to RFID data, focusing on supply chain object tracking through probabilistic models. While it shares similarities in data compression, SPIRE operates offline and concentrates on location inference rather than continuous traffic monitoring. Our system extends the concept of stream compression to dynamic road networks, emphasizing live traffic pattern detection without relying on historical object relationships or containment hierarchies.

The system by Y. Zhang et al. [7] presents an infrastructure-heavy solution that integrates WSN and RFID for traffic monitoring, relying on large-scale sensor deployments and fixed RFID readers. This contrasts with our lightweight, decentralized approach that reduces dependency on extensive infrastructure by compressing data at the source and tolerating data collisions. By minimizing the need for fixed installations, our sys-

tem achieves greater scalability and cost-efficiency, particularly in resource-constrained environments.

Zhuang et al. [8] use centralized map-matching to aggregate vehicle tracking data and identify traffic hot spots. This approach demands significant computational resources and relies on centralized data fusion, creating potential communication bottlenecks. Our system circumvents these limitations by distributing the detection process across WSN and RFID nodes, enabling localized data compression and reducing the need for centralized analysis. This decentralized structure enhances scalability and allows for continuous, resilient traffic monitoring.

Several works [9, 10, 11] propose advanced deep learning models such as graph neural networks, autoencoders, and spatio-temporal learning to estimate traffic volumes. These models address underdetermined and nonequilibrium flows, achieving high accuracy even with limited sensor coverage. However, they rely heavily on historical data, feature extraction, and extensive training, which introduces computational overhead. Our approach diverges by employing MinHashing for lightweight, real-time compression and analysis, providing immediate traffic insights without the need for large datasets, complex training pipelines, or deep learning frameworks.

Sambana and Srisudha [12] focus on post-incident detection, identifying accidents through IoT sensors and alerting emergency services. While useful for emergency response, this event-driven system contrasts with our continuous traffic monitoring approach, which can also achieve similar results by incorporating minimal historical data. By observing anomalies, such as increased traffic on typically low-volume roads and comparing them with consistently high-traffic edges, our system can infer accidents or disruptions. This method not only aids in identifying incidents but also highlights root causes by detecting shifts in traffic patterns. Additionally, our approach eliminates the need for extensive infrastructure such as PDA devices on installed apps. By relying solely on passive RFID tags on vehicles, the entire process occurs independently ensuring streamlined, low-cost monitoring without additional contributions from drivers or complex sensor networks.

Our method overcomes these limitations by adapting to the dynamic nature of traffic data and achieving scalability through the use of the MinHashing technique [21]. A central aspect of our approach is the innovative use of MinHash signatures, which are derived from the Size-Estimation Framework [20].

Chapter 3

Model Overview

The RFID sensor model operates within a hierarchical four-tier architecture, consisting of a Top Leader node, Regional Leader nodes, RFID-enabled motes, and passive RFID tags on vehicles. RFID motes are embedded along road edges and equipped with RFID readers to detect vehicle tags as they pass by.

3.1 Symbol Table Chapter 3

Symbol	Meaning	Notes
$\mathbf{RegL_j}$	Regional Leader node	All three notations appear in the text to refer to the same concept.
$\mathbf{R_g}$	Polygonal region covered by a particular Regional Leader	Non-overlapping region, one per leader.
$\mathbf{S_{e_i}}$	RFID sensor node (reader) at edge e_i	Collects tagIDs of passing vehicles.
$\mathbf{e_i}$	A road-network edge (road fragment) indexed by i	Each sensor is associated with one edge.
$\mathbf{T}$	Time window size for each scan	End of window triggers data transmission.
$\mathbf{O_{e_i}}$	Set of object tagIDs detected on edge e_i within time T	E.g., $O_{e_i} = \{\text{tagID}_1, \text{tagID}_2, \ldots\}$.
tagID	Unique identifier for each vehicle's RFID tag	Stored and transmitted by the sensors.
HoMoEdge	Hot Motion Edge	A single segment "hot path".
HoMoPath	Hot Motion Path	A combination of "hot edges".
L, **N**, **M**	Path-length indicators or maximum path length in tables	E.g., "$L > 1$" for paths of length 2 or more.
TopLeader	Central (top-tier) leader	Integrates data from all Regional Leaders.

Table 3.1: Symbol table for Chapter 3.

3.2 Architecture Overview

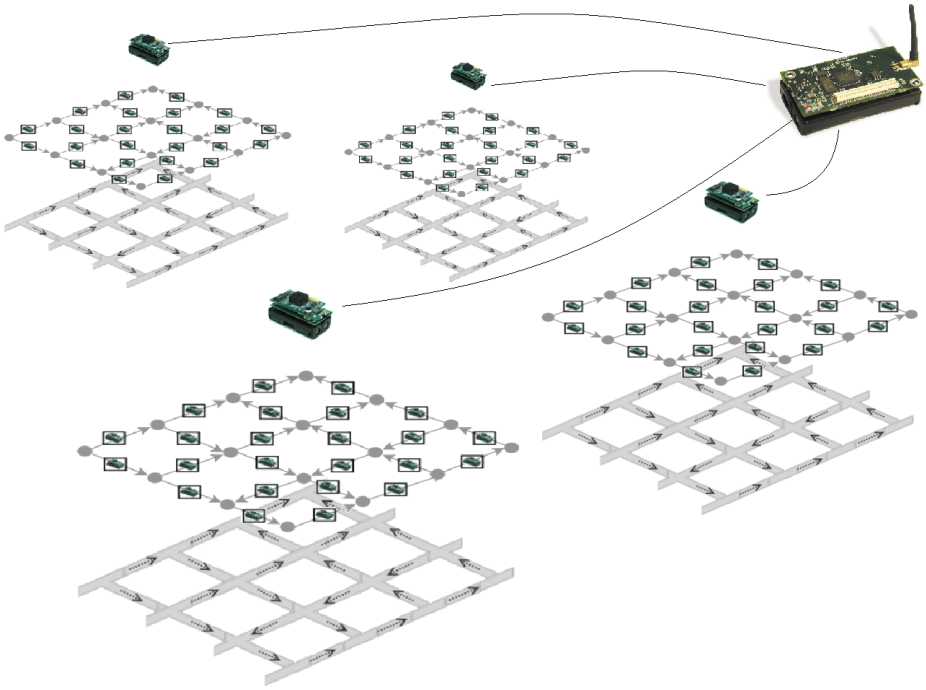


Figure 3.1: Architecture Overview.

Level 1 - RFID Tags: In this tier, RFID tags are attached to vehicles moving through the network. The key characteristic of this tier's participants is their lack of communication capabilities among each other. Their primary role is to respond to queries from the RFID readers in the tier above as they traverse different roads. Each moving object has a unique and distinct tagID.

Level 2 - RFID Readers: Embedded RFID-Readers in sensor nodes ,and each sensor represents a road edge , not necessary the whole road segment. The RFID-Reader emits a radio frequency (RF) signal, which provides the necessary energy to power up the passive tag. Each sensor(RFID Reader) node in this tier is associated with a specific Regional Leader.

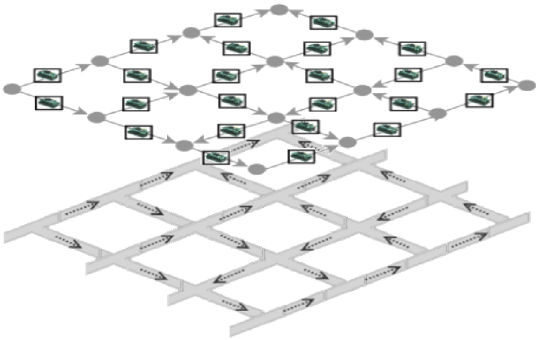


Figure 3.2: Level 2.

Level 3 - Regional Leader: Comprised of Leader Nodes (denoted as $RegL_j$), each covering a polygon region R_g. These regions, potentially of various shapes, do not overlap. Reg.Leader nodes are initialized with knowledge of the road network graph, enabling them to manage data and communications efficiently within their respective regions.

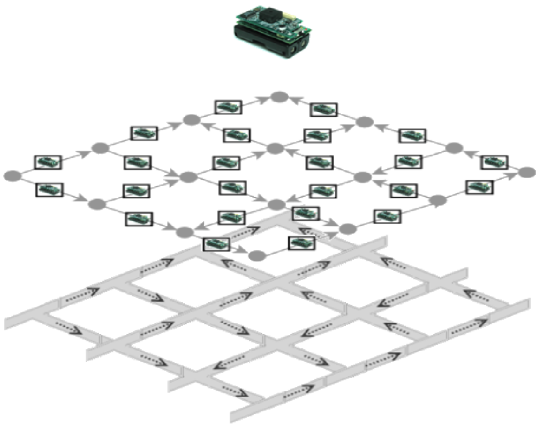


Figure 3.3: Level 3.

Level 4 - Top Leader: Acts as the central hub for information gathered by Leader Nodes in the Top Tier. Here, leaders from different regions share information on detected hot motion paths, enabling the Top Leader to analyze and identify path continuities across regions.

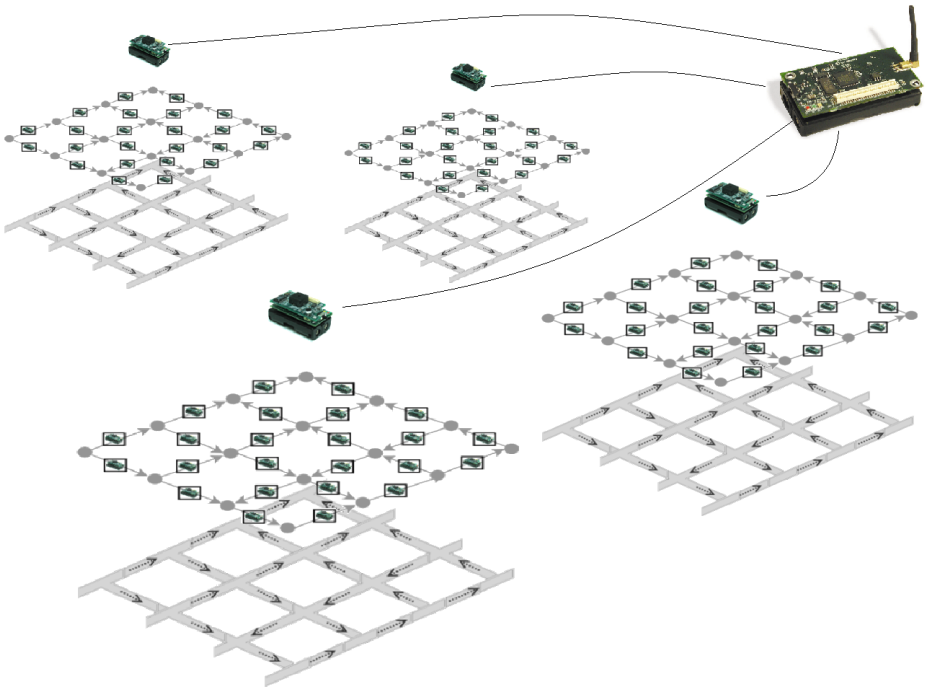


Figure 3.4: Level 4.

3.3 Data Framework Overview

With the sensor network model established and the architecture defined, it is essential to detail the organization and flow of data across the various tiers. This section focuses on how RFID scan data is accumulated, processed, and transmitted through the different levels of the communication hierarchy. We aim to illustrate the flow of information between sensors, Regional Leaders, and the Top Leader, highlighting the roles and responsibilities of each layer in facilitating efficient hot motion path detection. By examining the interactions within and across these layers, we provide a comprehensive understanding of how the data framework supports the overall system objectives.

3.3.1 Data Collection and Communication

RFID readers (S_{e_i}) scan tags of objects crossing their corresponding road network edge e_i, accumulating identifiers within a time window of size T. This collected set of object-tag IDs by S_{e_i} is denoted as $O_{e_i} = \{\text{tagID}_1, \text{tagID}_2, \ldots\}$. At the end of each time window, the collected data O_{e_i} is transmitted to the leader node covering the region that includes edge e_i.

Regional Leader Node($RegL_j$)	**Data Set** ($O_{e_{i..n}}$)
e_i	(tagID_1)
$\vdots$	$\vdots$
e_h	$(\text{tagID}_1, \text{tagID}_2)$
$\vdots$	$\vdots$
e_k	$(\text{tagID}_1, \text{tagID}_2, \ldots, \text{tagID}_{10})$
$\vdots$	$\vdots$
e_l	$(\text{tagID}_1, \text{tagID}_2, \ldots, \text{tagID}_{100})$
$\vdots$	$\vdots$
e_m	$(\text{tagID}_1, \text{tagID}_2, \ldots, \text{tagID}_{1000})$
$\vdots$	$\vdots$
e_n	$(\text{tagID}_1, \text{tagID}_2, \ldots, \text{tagID}_x)$
$\vdots$	$\vdots$
e_y	$(\text{tagID}_1, \text{tagID}_2)$
$\vdots$	$\vdots$
e_z	(tagID_1)
$\vdots$	$\vdots$

Table 3.2: Exemplary Data Organization at the Regional Leader Tier

Since the sensors are time-based, the size of O_{e_i} can vary significantly across different edges and time windows due to variations in traffic density. This variability highlights the potential discrepancy in the size of the data per edge and Regional Leader node, as the concept of compression becomes more complex when the raw data are not fixed-length. Various approaches may consider different criteria for determining what constitutes valid effective compression in such a dataset, requiring careful consideration to balance efficiency and accuracy.

Our approach is to treat the total size of all collected sets as a unified "black box", focusing on reducing its overall size regardless of how much each individual edge contributes. By abstracting the system in this way, we aim to optimize compression at a global scale, without being constrained by the variability of individual edge contributions. This approach allows for scalability and simplicity, while still preserving the core data needed for downstream analysis.

3.3.2 Data Organization at Regional Leaders - Hot Motion Path Extraction

Upon reception, the sets O_{e_i} are organized in a tabular format at each leader node, effectively fragmenting the traffic data collection horizontally across leaders. This fragmentation supports the localized extraction of hot motion paths, with potential for extending analysis to neighboring regions if paths intersect regional boundaries.

$(RegL_A)$	**Data Set** (O_{e_i})
e_i	(Set of tagIDs)
⋮	⋮
e_h	(Set of tagIDs)
⋮	⋮
e_k	(Set of tagIDs)
⋮	⋮
e_l	(Set of tagIDs)

$(RegL_B)$	**Data Set** (O_{e_o})
e_o	(Set of tagIDs)
⋮	⋮
e_p	(Set of tagIDs)
⋮	⋮
e_q	(Set of tagIDs)
⋮	⋮
e_r	(Set of tagIDs)

Table 3.3: Exemplary Data Organization at the Regional Leader Tier

Each regional leader is responsible for analyzing the edges within its designated area and identifying both HoMoEdges and HoMoPaths. HoMoEdges represent individual HOT road segments, while HoMoPaths are formed by combining these edges into meaningful sequences based on traffic flow.

RegLeader	Length	Details
RegL_A	1	HoMoEdges not belonging to $L > 1$ paths.
	2	HoMoPaths not belonging to $L > 2$ paths.
	3	HoMoPaths not belonging to $L > 3$ paths.
	⋮	⋮
	N	HoMoPaths not belonging to $L > N + 1$ paths.
⋮	⋮	⋮
RegL_B	1	HoMoEdges not belonging to $L > 1$ paths.
	2	HoMoPaths not belonging to $L > 2$ paths.
	3	HoMoPaths not belonging to $L > 3$ paths.
	⋮	⋮
	M	HoMoPaths not belonging to $L > M + 1$ paths.

Table 3.4: Identified HoMoEdges and HoMoPaths for Regional Leaders

3.3.3 Top Leader and Inter-Region Spanning Paths

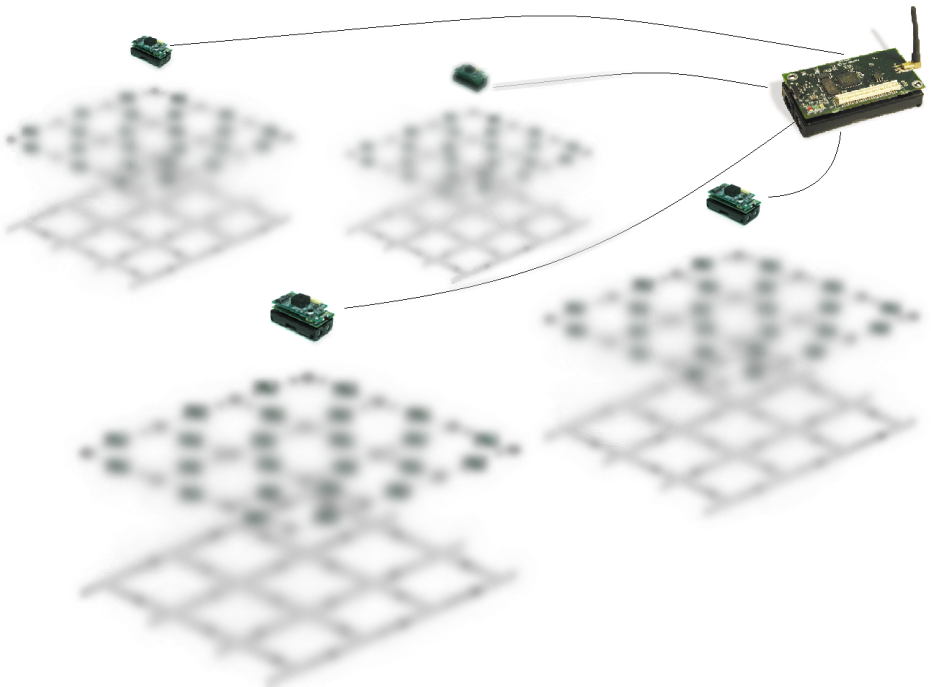


Figure 3.5: Top Leader - Region Leaders Communication

The Top Leader (TopLeader) is the central node responsible for integrating data from multiple Regional Leaders (RegLeaders) to identify spanning HoMoPaths that extend across regional boundaries. While RegLeaders independently analyze their local HoMoEdges and HoMoPaths, they do not exchange information directly with one another. Instead, the TopLeader collects summaries of detected HoMoPaths and boundary-specific edges from each RegLeader. By limiting communication to boundary-level data, this approach reduces the overhead of transmitting complete datasets while preserving the ability to construct meaningful global paths.

The TopLeader processes the incoming data to determine whether any HoMoPaths terminate at regional boundaries and, if so, checks for continuity with paths from adjacent regions. This process involves combining paths into spanning structures that link multiple regions. The summaries from RegLeaders can be transmitted in formats such as JSON, escaped JSON, or raw data, ensuring compatibility and efficiency. This hierarchical organization allows the TopLeader to focus on the inter-region connections without requiring full network knowledge.

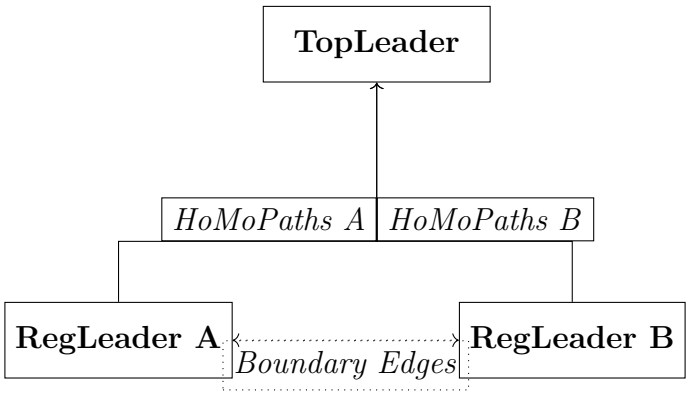


Figure 3.6: Communication between TopLeader and Regional Leaders for Spanning HoMoPaths

The figure above illustrates how the TopLeader consolidates data from RegLeaders to reconstruct spanning HoMoPaths. By focusing on boundary edges and inter-region continuity, the TopLeader ensures efficient detection of large-scale motion patterns without requiring exhaustive communication. This method minimizes redundancy, reduces data transfer costs, and provides a scalable framework for detecting hot motion paths in complex, distributed sensor networks. The hierarchical separation of local and global responsibilities highlights the adaptability of the system to varying traffic conditions and regional densities.

Chapter 4

Problem Formulation - HoMoPaD

In this chapter, we formalize the problem of Hot Motion Paths Detection (HoMoPaD) in a distributed manner across a road network. We begin by modeling the network as a directed graph, then define the criteria for paths to qualify as hot motion paths, provide formal definitions, and discuss the associated fundamental challenges.

4.1 Symbol Table Chapter 4

Symbol / Notation	Meaning	Notes
HoMoPaD	Hot Motion Paths Detection	Name of the overall problem.
$\mathbf{G} = (\mathbf{X}, \mathbf{E})$	Road network as a directed graph	X are vertices, E are edges.
$\mathbf{x_i} \in \mathbf{X}$	A vertex	Road intersection.
$\mathbf{e_i} \in \mathbf{E}$	A directed edge	Road segment.
$\mathbf{G_j} = (\mathbf{X_j}, \mathbf{E_j})$	Fragment of the global graph G	Associated with a single RegL_j.
$\mathbf{RegL_j}$	Regional Leader	Oversees region R_{g_j}.
$\mathbf{R_{g_j}}$	Region covered by a RegL_j	R_{g_j} is specifically the j-th region.
$\mathbf{O_{e_i}}$	Set of objects on edge e_i	–
$\mathbf{O_{R_g}}$	Union of objects in region R_g	$O_{R_g} = \bigcup_{e \in R_g} O_e$.
$\mathbf{P_i} = \langle \mathbf{e_s}, \ldots, \mathbf{e_f} \rangle$	A path in the directed graph	Sequence of edges from e_s to e_f.
$\text{Intersects}(P_i, R_g)$ true	Path P_i traverses region R_g	Used when calculating $\bigcup_{\forall R_g}$.
$\mathbf{T}$	Time window for sensor scans	Defines the reporting interval.
minHoMoSup	Threshold % for "hot" edges-paths	E.g., 20% means $\geq 20\%$ of local traffic.
$\mathbf{HoMoSup(e_i)}$	Hot motion support for edge e_i	$= \dfrac{\lvert O_{e_i} \rvert}{\lvert \bigcup O_{R_g} \rvert}$.
$\mathbf{HoMoSup(P_i)}$	Hot motion support for path P_i	$= \dfrac{\lvert O_{e_s} \cap \cdots \cap O_{e_f} \rvert}{\lvert \bigcup_{\text{Intersects}(P_i, R_g)} O_{R_g} \rvert}$.
HoMoEdge	Hot Motion Edge	A single segment "hot path".
HoMoPath	Hot Motion Path	A combination of "hot edges".
tagID	Unique identifier for each vehicle's RFID tag	Stored and transmitted by the sensors.

Table 4.1: Symbol table for Chapter 4.

4.2 Network Representation as a Directed Graph

We model the underlying road network as a directed graph $G = (X, E)$, where:

- **Vertices** $x_i \in X$: Each vertex represents a road intersection or junction.
- **Directed Edges** $e_i \in E$: Each edge corresponds to a road segment.
- **Bidirectional Segments**: Road segments allowing travel in both directions are represented by pairs of edges with opposite directions.

The graph G is fragmented across leader nodes to manage different regions:

- **Fragments** $G_j = (X_j, E_j)$: Each fragment is associated with a Regional Leader node $RegL_j$ that covers a specific region R_{g_j}.
- **Shared Vertices**: Fragments G_j may share vertices, indicating intersections relevant to multiple regions.
- **Unique Edges**: Each edge $e \in E$ is unique to a single fragment, ensuring that every road segment is managed by a single $RegL_j$.

4.3 Qualification Criteria for HoMoPaths

Users can pose continuous queries to the sensor network. This query establishes criteria that paths must meet to be considered significant, or 'hot'.

Query:

> "Report paths that are preferred by at least minHoMoSup% of the objects moving in the area, every T time units."

Our threshold minHoMoSup% is a percentage indicating the minimum proportion of traffic a path must have relative to the total traffic in its area to be reported as hot. Considering that, it quantifies the popularity or preference of a path among moving objects. Principally, it is calculated as the ratio of the traffic on the path's edges to the total traffic within the regions it spans, excluding other areas of the road network. Furthermore the Time Units (T), presents the frequency at which the RFID sensors will scan their areas.

4.3.1 Formal Definition of Hot Motion Paths

We formalize the concept of hot motion paths within a directed graph representing a road network, based on two pivotal criteria.

Definition
A path $P_i = \langle e_s, \ldots, e_f \rangle$ is considered a hot motion path during a time window T if it satisfies both of the following conditions:

1. **Condition Minimum Support :**

$$HoMoSup(P_i) = \frac{\left|O_{e_s} \cap \ldots \cap O_{e_f}\right|}{\left|\bigcup_{\forall R_g:\text{Intersects}(P_i,R_g)=\text{true}} O_{R_g}\right|} \geq \text{minHoMoSup} \tag{4.1}$$

   This ensures that the path has a significant level of traffic compared to the total traffic in the intersected regions.

   - $(O_{e_s} \cap \ldots \cap O_{e_f})$: Represents the set of objects that traverse every edge in the path from e_s to e_f.
   - $\left(\bigcup_{\forall R_g:\text{Intersects}(P_i,R_g)=\text{true}} O_{R_g}\right)$: Includes all objects within the regions R_g intersected by the path P_i.
   - (minHoMoSup) : Threshold indicating the path's traffic must be a significant proportion of the total traffic in intersected regions.

2. **Condition Maximality:**

$$\nexists P_j \text{ such that } HoMoSup(P_j) \geq \text{minHoMoSup} \wedge \text{Contains}(P_j, P_i) = \text{true}, \forall i \neq j \tag{4.2}$$

   This condition ensures that no sub-path of a hot motion path should be reported if the larger path containing it also meets the minHoMoSup threshold. This criterion emphasizes the need for concise reporting, ensuring that results focus on the longest and most comprehensive paths that meet the criteria.

Summarizing, hot motion path represents a frequently used route within a road network during a given time window. For a path to be considered "hot", it must meet two main conditions. First, a significant number of moving objects must traverse every edge of the path compared to the total traffic in the regions it passes through. Second, the path must be the longest such route, meaning we do not consider shorter segments if they are already part of a longer, valid hot motion path. These criteria help us focus on the most relevant and informative traffic patterns within the network.

4.4 HoMoEdges - HoMoPaths Detection

For an edge e_i, the hot motion support is the proportion of objects that have crossed this edge relative to the total number of objects detected in the region it intersects:

$$HoMoSup(e_i) = \frac{|O_{e_i}|}{\left|\bigcup_{\forall R_g:\text{Intersects}(e_i,R_g)=\text{true}} O_{R_g}\right|} \tag{4.3}$$

Since an edge is a single segment, the numerator simplifies to $|O_{e_i}|$, and the denominator accounts for all objects in the region intersected by the edge.

On the other hand, for a path to be defined as a hot motion path we need the intersection of the objects on the edges, as described previously in 4.1,

$$HoMoSup(P_i) = \frac{\left|O_{e_s} \cap \ldots \cap O_{e_f}\right|}{\left|\bigcup_{\forall R_g:\text{Intersects}(P_i,R_g)=\text{true}} O_{R_g}\right|} \tag{4.4}$$

When evaluating paths, we consider both the intersection of objects across all edges in the path and the union of objects in all regions intersected by the path. Normally, the constituent edges of a hot motion path must first be identified as hot motion edges. This ensures that a path's significance is inherently linked to the significance of its segments, adhering to the principle of transitive closure.

This ratio captures the relative significance of a path within the broader context of network traffic, requiring both specific traffic on the path and the general traffic context of the intersected regions.

4.4.1 Simplified Example: Hot Motion Edges vs. Hot Motion Paths

Consider a region R_g with four edges: e_1, e_2, e_3, and e_4. Assume a distribution of objects with explicit IDs as follows:

- $O_{e_1} = \{o_1, o_2, o_3, o_4, o_5\}$, $|O_{e_1}| = 5$
- $O_{e_2} = \{o_3, o_4, o_5, o_6\}$, $|O_{e_2}| = 4$
- $O_{e_3} = \{o_7, o_8, o_9\}$, $|O_{e_3}| = 3$
- $O_{e_4} = \{o_{10}\}$, $|O_{e_4}| = 1$
- The union of all objects in R_g:

$$O_{R_g} = O_{e_1} \cup O_{e_2} \cup O_{e_3} \cup O_{e_4} = \{o_1, o_2, o_3, o_4, o_5, o_6, o_7, o_8, o_9, o_{10}\}, \quad \left|O_{R_g}\right| = 10$$

Assume minHoMoSup is 20%.

Hot Motion Edges

Calculate the hot motion support for each edge, based on (4.3):

$$HoMoSup(e_i) = \frac{|O_{e_i}|}{|O_{R_g}|}$$

- $HoMoSup(e_1) = \frac{5}{10} = 0.5 > 0.2$
- $HoMoSup(e_2) = \frac{4}{10} = 0.4 > 0.2$
- $HoMoSup(e_3) = \frac{3}{10} = 0.3 > 0.2$
- $HoMoSup(e_4) = \frac{1}{10} = 0.1 < 0.2$

Edges e_1, e_2, and e_3 qualify as hot motion edges.

Hot Motion Paths

Next, we evaluate possible paths composed of hot motion edges, based on (4.4):.

Path $P_{12} = \langle e_1, e_2 \rangle$

$$O_{e_1} \cap O_{e_2} = \{o_3, o_4, o_5\}, \quad |O_{e_1} \cap O_{e_2}| = 3$$

$$HoMoSup(P_{12}) = \frac{3}{10} = 0.3 > 0.2$$

Path $P_{13} = \langle e_1, e_3 \rangle$

$$O_{e_1} \cap O_{e_3} = \emptyset, \quad |O_{e_1} \cap O_{e_3}| = 0$$

$$HoMoSup(P_{13}) = \frac{0}{10} = 0 < 0.2$$

- Edges e_1, e_2, and e_3 are HoMoEdges and the path $P_{12} = \langle e_1, e_2 \rangle$ is a HoMoPath.
- P_{12} is reported as a hot motion path. Thus, according to (4.2), the edges e_1 and e_2 are not reported separately as hot motion edges because they are part of the maximal hot motion path P_{12}.
- Edge e_3 remains a hot motion edge since it does not form a hot motion path with other edges.

This conclusion highlights how the integration of multiple edges based on their traffic volume can alter the landscape of hot motion paths within a network.

4.4.2 Strategic Transmission of Sensor Data

The complexity of identifying hot motion paths can be dissected into three critical areas:

1. **Identifying Hot Motion Edges**:
   - Requires understanding the count of unique object IDs for each edge and the total in the network.
   - Challenges include efficiently managing and querying large datasets.
2. **Identifying Hot Motion Paths**:
   - Involves comparing sets of object IDs to identify intersections.
   - Challenges include the computational cost of set operations and scalability.
3. **Data Compression and Optimization**:
   - Reducing sensor data volume is essential for efficient transmission, faster processing, and lower resource demand.
   - Challenges include designing efficient data compression techniques without compromising accuracy.

This discussion highlights the multifaceted nature of the problem, encompassing computational, methodological, and technological challenges in real-time traffic analysis.

4.5 Compression Definition and Challenges

Compression in Sensor Networks:

Defining compression in our model, as described in subsection 3.3.1, poses unique challenges due to non-uniform traffic volumes. Each sensor records tagIDs at different rates, influenced by local traffic conditions. Uniformly compressing such datasets demands a consistent approach that accommodates these variations while minimizing communication costs.

Proposed Compression Approach:

To address this, we treat the entire dataset as a single *black box* of size N bytes. Compression is applied globally, with all sensors min-hashing their data using the same seed and hash functions. The compressed size of the dataset is defined as:

$$\text{Compressed Size} = \frac{N}{X}, \quad \text{where } X \text{ is the compression ratio.}$$

Chapter 5

Compression via Minhashing

In this section, we present our approach to identifying hot motion edges and paths using MinHash signatures. MinHashing, traditionally known for estimating the similarity between sets, plays a pivotal role in our methodology by enabling efficient data compression, representation and comparison. Our goal is to leverage MinHash signatures for detecting hot motion paths in the distributed regions which also may span across regional boundaries. The strengths of MinHashing combined with our hierarchical framework, minimize communication overhead and maintain the accuracy required for HoMoPaD.

5.1 Symbol Table Chapter 5

Symbol	Meaning	Notes
$\mathbf{h_i}$	i-th hash function in a family	Maps set elements uniformly into $[0, H]$
$\mathbf{k}$	Number of independent hash functions	–
$\mathbf{H}$	Hash-value range upper bound	$H = 1$ (normalized), or e.g., $[0, 2^{bits} - 1]$
$\mathbf{n}$	Size $\lvert S \rvert$ of a set	–
$\mathbf{m_i}$	Minimum hash value under the i-th hash function	–
$\overline{\mathbf{m}}$	Empirical average of the k minimum-hash values	$\overline{m} = \frac{1}{k} \sum_{i=1}^{k} m_i$
A-LESE	Atomic Least Element Size Estimation	Cohen's Estimation $\lvert S \rvert$ via MinHash mins
G-LESE	Global Least Element Size Estimation	Estimates $\left\lvert \bigcup_i S_i \right\rvert$ via merged min-hash vectors
JISE	Jaccard Index Size Estimation	Iteratively estimates $\left\lvert \bigcup_i S_i \right\rvert$ using pairwise Jaccard
CCM	Counter Cardinality MinHash	Sensors transmit {`Counter`, MinHash}; local size is exact
FCM	Full-stack Cardinality MinHash	Sensors transmit MinHash only; local sizes also estimated
HoMoEdge	Hot Motion Edge	A single segment "hot path"
HoMoPath	Hot Motion Path	A combination of "hot edges"
$\boldsymbol{\varepsilon}$, $\boldsymbol{\delta}$	Error tolerance, confidence level	Used in Hoeffding's bounds
$\boldsymbol{\Delta}$	Error gap between two JISE estimates	Larger Δ indicates greater overlap

Table 5.1: Symbol table for Chapter 5.

5.2 MinHashing

A hash function h is applied to each element of a set, producing a hash value for each element. The MinHash $h_{\min}(O_{e_i})$ of a set O_{e_i} , is the minimum of these hash values:

$$h_{\min}(O_{e_i}) = \min\{h(o_{e_i}) \mid o_{e_i} \in O_{e_i}\} \tag{5.1}$$

Consider a set $O_{e_i} = \{\text{tagID}_1, \text{tagID}_2, ..., \text{tagID}_n\}$, where tagID_n represents the n-th element of the set. To compute the MinHash value of O_{e_i}, we apply a hash function h to each element of O_{e_i}, yielding a hash value for each element:

$$\begin{aligned} h(\text{tagID}_1) &= h_1 \\ h(\text{tagID}_2) &= h_2 \\ &\vdots \\ h(\text{tagID}_n) &= h_n \end{aligned} \tag{5.2}$$

The MinHash of the set O_{e_i}, is :

$$h_{\min}(O_{e_i}) = \min\{h_1, h_2, ..., h_n\} \tag{5.3}$$

where h_n is the hash value of the n-th element $tagID_n$ under the hash function h.

5.2.1 Jaccard Similarity

Jaccard similarity measures the similarity between finite sample sets, defined as the size of the intersection divided by the size of the union of the sets. It is widely used in applications requiring comparisons of sets, such as clustering, classification, or measuring overlap in data streams. Mathematically, it is expressed as follows:

Given two sets O_{e_i} and O_{e_j}, with $O_{e_i} \neq \emptyset$ or $O_{e_j} \neq \emptyset$, in a region $R_g = O_{e_i} \cup O_{e_j}$:

$$J(O_{e_i}, O_{e_j}) = \frac{|O_{e_i} \cap O_{e_j}|}{|O_{e_i} \cup O_{e_j}|} = \frac{\left|O_{e_s} \cap O_{e_f}\right|}{\left|\bigcup_{\forall R_g:\text{Intersects}(P_i,R_g)=\text{true}} O_{R_g}\right|}.$$

$$\text{where } 0 \leq J(O_{e_i}, O_{e_j}) \leq 1,$$

where a Jaccard similarity of 1 indicates identical sets, while a similarity of 0 indicates no overlap. This metric is particularly useful in regions where multiple sets contribute to the union, allowing an aggregated view of overlap across spatial or temporal dimensions.

Example

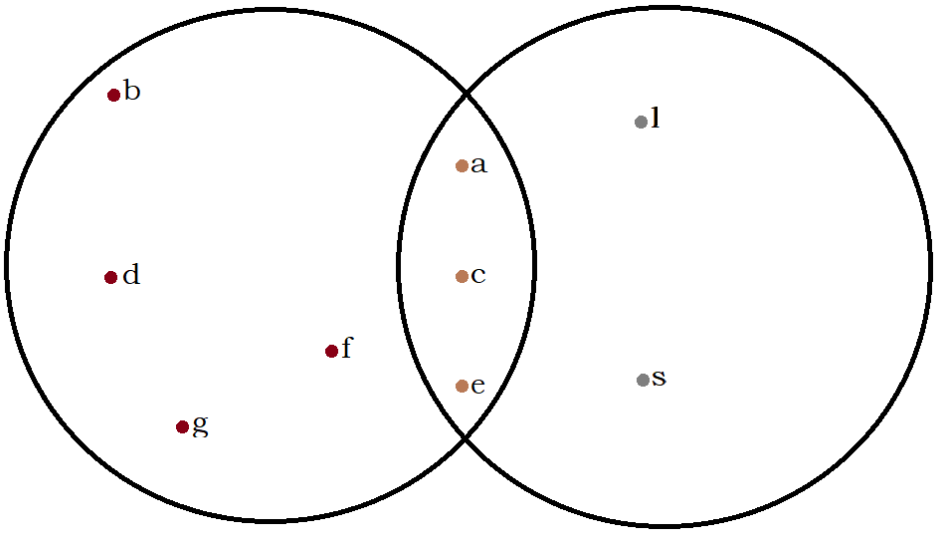


Figure 5.1: Example Jaccard Similarity.

Consider two sets:

$$O_{e_i} = \{a, b, c, d, e, f, g\}, \quad O_{e_j} = \{a, c, e, l, s\}.$$

The intersection of these sets are:

$$O_{e_i} \cap O_{e_j} = \{a, c, e\}$$

The union of these sets are:

$$O_{e_i} \cup O_{e_j} = \{a, b, c, d, e, f, g, l, s\}.$$

The Jaccard similarity is then calculated as:

$$J(O_{e_i}, O_{e_j}) = \frac{|O_{e_i} \cap O_{e_j}|}{|O_{e_i} \cup O_{e_j}|}$$

$$= \frac{|\{a, c, e\}|}{|\{a, b, c, d, e, f, g, l, s\}|} = \frac{3}{9} = \frac{1}{3}.$$

This example illustrates how Jaccard similarity effectively quantifies the overlap between two sets, independent of their absolute sizes. Its utility becomes particularly significant when paired with methods like *MinHashing*, which facilitate efficient approximations of Jaccard similarity. Furthermore, the connection between Jaccard similarity and our Hot Motion Path Detection logic, as defined in Section 4.4 under Condition 4.1 and 4.4, becomes evident, reinforcing the foundational role of this metric in our approach.

5.3 MinHash Values and Jaccard Similarity

MinHash relies on the principle that the probability of a hash collision for a pair of elements correlates with their Jaccard similarity [31]. Formally, for any two sets O_{e_i} and O_{e_j}, the Jaccard similarity $J(O_{e_i}, O_{e_j})$ is equal to the probability that the MinHash of O_{e_i} is equal to the MinHash of O_{e_j}.

Consider two sets of edges or sensors, O_{e_i} and O_{e_j}, each associated with a series of MinHash values extracted through k random hash functions $\pi_1, \pi_2, \ldots, \pi_k$. The MinHash values for O_{e_i} and O_{e_j} can be represented as follows:

$$\begin{aligned} &\{\min(\pi_1(O_{e_i})), \min(\pi_2(O_{e_i})), \ldots, \min(\pi_k(O_{e_i}))\}, \\ &\{\min(\pi_1(O_{e_j})), \min(\pi_2(O_{e_j})), \ldots, \min(\pi_k(O_{e_j}))\} \end{aligned} \tag{5.4}$$

The Jaccard similarity estimate, $\widehat{J}(O_{e_i}, O_{e_j})$, consists of k random variables $Y_1, Y_2, \ldots, Y_k$, where each Y_λ is defined as:

$$Y_\lambda = \begin{cases} 1 & \text{if } \min(\pi_\lambda(O_{e_i})) = \min(\pi_\lambda(O_{e_j})), \\ \emptyset & \text{otherwise,} \end{cases} \tag{5.5}$$

for $\lambda = 1, 2, \ldots, k$.

Therefore, the Jaccard similarity estimate can be mathematically expressed as:

$$\widehat{J}(O_{e_i}, O_{e_j}) = \frac{1}{k} \sum_{\lambda=1}^{k} \mathbf{1}_{\{\min(\pi_\lambda(O_{e_i}))=\min(\pi_\lambda(O_{e_j}))\}} = J(O_{e_i}, O_{e_j}) \tag{5.6}$$

Where $J(O_{e_i}, O_{e_j})$ is the expectation and $\mathbf{1}$ denotes the indicator function, it is used to construct the $\widehat{J}(O_{e_i}, O_{e_j})$, which is computed as the sum of indicator variables Y_λ over all k hash functions, normalized by k. Formally, this can be expressed as:

$$\widehat{J}(O_{e_i}, O_{e_j}) = \frac{1}{k} \sum_{\lambda=1}^{k} Y_\lambda = J(O_{e_i}, O_{e_j}) \tag{5.7}$$

Proof. □

Let $d = |O_{e_i} \cap O_{e_j}|$ and $m = |O_{e_i} \cup O_{e_j}|$. The probability that the first element of a random / hash function of $O_{e_i} \cup O_{e_j}$ comes from $O_{e_i} \cap O_{e_j}$ is d/m, which is exactly the Jaccard similarity $J(O_{e_i}, O_{e_j})$. This follows because, under a random permutation /hash function, each element in $O_{e_i} \cup O_{e_j}$ has an equal chance of being first, making the probability of the first element being in both O_{e_i} and O_{e_j} equal to the ratio of the intersection to the union of the two sets.

Assuming k independent hash functions and a scenario where the hash functions behave ideally (i.e., the probability of a hash function agreeing on both sets equals the Jaccard similarity), the expectation $\mathbb{E}[\widehat{J}(O_{e_i}, O_{e_j})]$ is $J(O_{e_i}, O_{e_j})$.

However, $\mathbb{E}[|\widehat{J}(O_{e_i}, O_{e_j}) - J(O_{e_i}, O_{e_j})|]$ captures the expected magnitude of deviation from the true similarity, which might be directly influenced by the number of hash functions k and the characteristics of the sets. The MinHash technique ensures that as k increases, the variance of $\widehat{J}(O_{e_i}, O_{e_j})$ decreases, thereby improving the accuracy of the Jaccard similarity estimation.

Hoeffding's inequality[26] is tailored for sums of bounded independent random variables, which is directly applicable to the MinHash technique for estimating the Jaccard Index.

Given the Hoeffding's inequality for the sum of independent bounded variables Y_i, where $a_i \leq Y_i \leq b_i$, we have:

$$\Pr\left(\left|\sum_{i=1}^{k} Y_i - \mathbb{E}\left[\sum_{i=1}^{k} Y_i\right]\right| \geq k\epsilon\right) \leq 2e^{-2k\epsilon^2/\sum_{i=1}^{k}(b_i-a_i)^2}. \tag{5.8}$$

To include the operation of multiplying by $1/k$ within the sums and the $k\epsilon$ term in the probability, and to explicitly state that $k > 0$, we proceed as follows:

$$\Pr\left(\left|\frac{1}{k}\sum_{i=1}^{k} Y_i - \mathbb{E}\left[\frac{1}{k}\sum_{i=1}^{k} Y_i\right]\right| \geq \epsilon\right) \leq 2e^{-2k\epsilon^2/\sum_{i=1}^{k}(b_i-a_i)^2}. \tag{5.9}$$

Considering the specific case of Jaccard similarity estimation via MinHash, where the range of each independent variable Y_i is $[0, 1]$, the Hoeffding's inequality simplifies as follows:

$$\Pr\left(\left|\frac{1}{k}\sum_{i=1}^{k} Y_i - \mathbb{E}\left[\frac{1}{k}\sum_{i=1}^{k} Y_i\right]\right| \geq \epsilon\right) \leq 2e^{-2k\epsilon^2}. \tag{5.10}$$

Which lead us based on :

$$\Pr\left(|\widehat{J}(O_{e_i}, O_{e_j}) - J(O_{e_i}, O_{e_j})| \geq \epsilon\right) \leq 2e^{-2k\epsilon^2}, \tag{5.11}$$

where ϵ represents the deviation from the true Jaccard Index $J(O_{e_i}, O_{e_j})$ and k represents the number of hash functions .

Setting the right side of this inequality to δ gives us:

$$\delta = 2e^{-2k\epsilon^2}. \tag{5.12}$$

Solving for k gives the number of hash functions needed to ensure that the estimated Jaccard similarity is within $\pm\epsilon$ of the true similarity with probability at least $1-\delta$. Using big-O notation, this can be expressed as:

$$k = O\left(\frac{\ln(2/\delta)}{2\epsilon^2}\right). \tag{5.13}$$

To save communication and achieve compression, the number of hash functions should generally satisfy $k < |O_{e_i}|$, where $|O_{e_i}|$ denotes the size of the transmitted object set. However, caution is required with the definition of compression, as outlined in 3.3.1, since the presence of time-based sensors can result in edge cases where k may approach or even exceed $|O_{e_i}|$, particularly for edges with very low traffic volumes.

5.3.1 Approximate Hot Paths via Local MinHash Jaccard

Recall that the hotness criterion for a path P_i in (4.4) compares

$$\frac{\left|\bigcap_{\text{edges in } P_i} O_e\right|}{\left|\bigcup_{R_g:\,\text{Intersects}(P_i,R_g)=\text{true}} O_{R_g}\right|} \quad \text{to some threshold} \quad \tau.$$

In some applications, we replace this *global* denominator with a *local* Jaccard fraction or an equivalent MinHash-based overlap among only the edges in the path. Concretely, one might approximate

$$\Big|\bigcap_{\text{edges in } P_i} O_e\Big| \Big/ k,$$

where k is the number of hash functions capturing the overlap from a local perspective. Below, we enumerate our predictions for the main theoretical consequences of this approach :

1) **No more False Negatives.** If the local Jaccard ratio $\frac{|A\cap B|}{|A\cup B|} \leq \tau$ fails for two sets (or for edges in a path), then using the larger global union $|U|$ (i.e., the full network union) can *only* reduce that fraction further. Consequently, a path that is *not* hot under the local denominator cannot become hot under the full global denominator. Hence, we introduce *no false negatives* by adopting a local approximation.

2) **Possible more False Positives.** Because $|A \cup B| \leq |U|$, it follows that

$$\frac{|A \cap B|}{|A \cup B|} \;\geq\; \frac{|A \cap B|}{|U|}.$$

Therefore, if local Jaccard suggests a path is hot (above τ), the global fraction $\frac{|A\cap B|}{|U|}$ might still fall below τ, yielding a false positive. This mismatch is most pronounced when the path's union is a small fraction of the full network.

3) **Empirical Observation at Higher Thresholds.** Interestingly, accuracy of this local approach *improves*. The reason is that only sets or paths with genuinely large intersections can pass a high threshold, and such sets already cover most of the relevant objects in the network. In that scenario, the difference between $|A \cup B|$ and the full union $|U|$ becomes negligible, so the MinHash approximation aligns more closely with the global fraction.

4) **When the Local Approximation Works Well.** In general, the local Jaccard estimate via MinHash serves as a reliable approximation for (4.4) under the following conditions:

   i) The path or edges under consideration already dominate the set of relevant objects (i.e., $|A \cup B| \approx |U|$).

   ii) The Region is of moderate size—not too large, but also not trivial—so that the global universe $|U|$ is not excessively large relative to $|A \cup B|$. This ensures that the union of the sets involved in the path is not dwarfed by the rest of the network, making accurate approximation feasible.

   iii) The remaining parts of the network contribute relatively few additional objects beyond those found in the hot edges.

   iv) The empirical thresholds are high enough that only truly large overlaps are retained.

   Under these conditions, the fraction $\frac{|A \cap B|}{|A \cup B|}$ remains very close to $\frac{|A \cap B|}{|U|}$, limiting false positives.

A key reason why accuracy improves at higher thresholds—despite the usual concerns about false positives—is that the paths, that surpass a high threshold tends to be naturally limited to those with substantial support. When a path or edge is required to meet an intersection-to-union ratio of, say, greater or equal than 40% or 50%, only the most dominant connections persist. These edges are easier to approximate using local Jaccard or MinHash because:

- Their intersection size is inherently large, reducing approximation errors.
- The discrepancy between local and global denominators diminishes as the edge becomes dominant.
- Moderate - Higher thresholds are filtering out noise from weak connections, reducing cumulative approximation discrepancies.

Thus, at moderate or higher thresholds, the local MinHash-based approach aligns more closely with the exact global measure, not due to an improvement in the algorithm itself, but because the population of tested edges becomes skewed towards trivially strong ones.

In summary, using the local MinHash overlap for hot-path detection yields *no false negatives* but can produce occasional false positives if the path's union is much smaller than the total network union. At moderate-higher thresholds—where only large intersections persist—this discrepancy becomes minimal, causing the local approach to track the full global fraction quite well.

This is acceptable in our case, as our primary interest lies in identifying hot motion paths, rather than just isolated hot edges. If a path consists of only 2–3 edges, its contribution and informational value are not particularly extraordinary. By emphasizing longer-stronger paths, we ensure that the detected patterns capture meaningful and sustained motion trends rather than transient connections.

Considering that, we propose the following HoMoPaD (Hot Motion Paths Detection algorithm)

Algorithm 1 HoMoPath Detection (HOMOPAD)

Require: (i)Edges-Connections, (ii)$HoMoEdges$ Dictionary , (iii)$Threshold$ ($minHoMoSup$), (iv)Minimum path length n

$Length \leftarrow n$
while `Boolean Matrix Contains Non-Zero Values` **do**
  $HotPaths \leftarrow$ Initialize_Empty_List()
  for all `Path` p `on Horizontal Row (e.g.,` $p = \{e_1, e_2, ..., e_{n-1}\}$`)` **do**
    for all `Edge` q `on Vertical Column` **do**
      if `p[len(p)] is Connected to` q **then**
        $JaccardIndex \leftarrow$ Calculate_Jaccard_Index($Signatures_p, Signatures_q$)
        if $JaccardIndex \geq Threshold$ **then**
          $HotPaths \leftarrow$ Add($p \cup q$, MinHash($Signatures_p \cap Signatures_q$))
        end if
      end if
    end for
  end for
  $HoMoPaths\_Dict[Length] \leftarrow HotPaths$
  Remove_Subsets($HoMoPaths\_Dict[Length]$)
  Update_Matrix($HoMoPaths\_Dict[Length]$, BooleanMatrix)
  $Length \leftarrow Length + 1$
end whilereturn $HoMoPaths\_Dict$

Explanation:

Inputs and Initialization: The algorithm requires a dictionary of $HoMoEdges$, which represents connections between edges in the network, and a $Threshold$, the minimum Jaccard similarity required to establish a connection. A Boolean matrix is used to track these connections and identify if a connection can be defined as Hot or not. The horizontal rows represents current paths (length 1-edges- ...n), and the vertical columns represents potential edges that can extend these Hot Motion Paths. The potential edges (columns) will be replaced by the HoMoEdges , after the HoMoEdge Phase.

Initial path length (**Line 1:**) The initial path length, $Length$, is set to $n = 2$, starting with extensions of single Hot Motion Edges.

Iterative Process (**Line 2:**) The main loop (`while Boolean Matrix Contains Non-Zero Values`) continues as long as the Boolean matrix contains at least one "1" value. The loop ends when there are no possible connections or the Boolean matrix has no "1" values left, indicating that no further connections or extensions can be made. In each iteration, the algorithm detects Hot Motion Paths of increasing lengths.

Initialization of HotPaths (**Line 3:**) A new list, `HotPaths`, is initialized to store the paths identified at the current length. Each entry in this list consists of two components: the first represents an edge (e.g., e_1) or a path (e.g., $e_1, e_2, ..., e_n$), while the second holds the associated objects that belong to the corresponding edge or path (e.g., $TagID_1, TagID_2, ..., TagID_m$). This structure ensures that both the connectivity information and the objects linked to each path are stored for further analysis.

Vertical and Horizontal Axis Logic (**Lines 4, 5, and 6:**) For every path p on the horizontal row, the algorithm examines edges q on the vertical column to determine if q can extend p, by principally checking if the last edge of p (`p[len(p)]`) is connected to q. Paths from the previous iteration populate the horizontal row, while the vertical column holds edges that might form new paths. Cyclic paths, such as $\{e_1, e_2, e_1\}$, are also excluded, as they do not contribute to meaningful HoMoPaths.

MinHashing and Jaccard Evaluation (**Line 7:**) If the last edge of p is connected to q,the Jaccard similarity for the MinHash signatures between p and q are calculated in order measure their overlap.

Connection and Jaccard Evaluation (**Lines 8, 9:**) If the similarity exceeds the $Threshold$, the algorithm extends p with q, and the combined path is added to `HotPaths`. For example, if $p = \{e_1, e_2\}$ and $q = e_3$, the resulting path becomes $\{e_1, e_2, e_3\}$.

Path Storage and Pruning (**Lines 14 and 15:**) Paths detected in the current iteration are stored in $HoMoPaths\_Dict[Length]$. To maintain efficiency, subsets of paths (paths entirely contained within longer paths) are removed using `Remove_Subsets`.

Matrix Update and Path Length Increment (**Lines 16 and 17:**) The Boolean matrix is updated to reflect newly formed connections, ensuring that only relevant edges and paths are considered in subsequent iterations. The path length, $Length$, is incremented to process longer paths in the next iteration.

Final Output (**Line 18:**) The algorithm returns $HoMoPaths\_Dict$, which groups detected paths by length and includes their associated object IDs.

How Matrices Work:

The Boolean matrix is a 2D structure where (i)Rows represent current paths (Horizontal Rows), (ii)Columns represent potential edges (Vertical Columns) and (iii)a value of "1" indicates that a given edge can extend the current path. For example, if the intersection of row p and column q is "1", the algorithm considers extending p with q. By iteratively populating and updating this matrix, the algorithm narrows down possible connections, focusing on meaningful extensions while ensuring computational efficiency.

5.3.2 Example Workflow

To illustrate the *HoMoPath Detection* algorithm, we begin with the following assumptions:

Boolean Matrix Identifying Hot Motion Edges

Boolean Matrix (HoMoEdge Phase) =

	e_1	e_2	e_3	e_4	e_5
e_1	1	0	0	0	0
e_2	0	1	0	0	0
e_3	0	0	1	0	0
e_4	0	0	0	0	0
e_5	0	0	0	0	1

The matrix above represents the output of the Hot Motion Edge Detection phase (explained in Section 5.4). It highlights which edges are identified as Hot Motion Edges. From the diagonal values, we can observe that all edges except e_4 are considered Hot Motion Edges.

Inputs and Initialization:

Based on the data and assuming the number of hash functions is 2, the inputs are as follows:

- `HoMoEdges`: Initial edges and their associated MinHash signatures:

$$\texttt{HoMoEdges} = \begin{aligned} &\{e_1\} : \text{MinHashSignature of } e_1 = \{1, 2\}, \\ &\{e_2\} : \text{MinHashSignature of } e_2 = \{2, 3\}, \\ &\{e_3\} : \text{MinHashSignature of } e_3 = \{3, 4\}, \\ &\{e_5\} : \text{MinHashSignature of } e_5 = \{3, 4\}. \end{aligned}$$

- Threshold: 0.2 (20% Jaccard Index).

- Initial Boolean Matrix:

Boolean Matrix (Initialization) =

	e_1	e_2	e_3	e_5
e_1	1	0	0	0
e_2	0	1	0	0
e_3	0	0	1	0

Rows will represent the HoMoPaths, which in this phase represent the current HoMoEdges as HoMoPaths of length 1. The columns represent the HoMoEdges that can extend the HoMoPaths(for this phase HoMoEdges). Note that e_5 is pruned as a horizontal row since it cannot lead to another HoMoEdge. Due to the transitive closure principle [28], the vertical columns exclusively represent HoMoEdges.

- Edges-Connections: Road connections are assumed as follows:

$$e_1 \to e_2, \quad e_2 \to e_3, \quad e_3 \to e_5.$$

Iteration : Detection of Hot Motion Paths, Length = 2

- Horizontal Rows (Current HoMoPaths of Length = 1 / HoMoEdges): $\{e_1, e_2, e_3\}$.
- Vertical Columns (Remaining HoMoEdges): $\{e_1, e_2, e_3, e_5\}$.
- The algorithm evaluates the Boolean matrix for potential connections. For example:

$$\widehat{J}(e_1, e_2) = \frac{|\{2\}|}{2} = \frac{1}{2} > 0.2$$

Boolean Matrix (Iteration 1) =

	e_1	e_2	e_3	e_5
e_1	X	1	0	0
e_2	0	X	0	0
e_3	0	0	X	0

- Path $\{e_1, e_2\}$ is added to `HotPaths`.

Similarly, paths $\{e_2, e_3\}$

Boolean Matrix (Iteration 1) =

	e_1	e_2	e_3	e_5
e_1	X	1	0	0
e_2	0	X	1	0
e_3	0	0	X	0

and $\{e_3, e_5\}$

Boolean Matrix (Iteration 1) =

	e_1	e_2	e_3	e_5
e_1	X	1	0	0
e_2	0	X	1	0
e_3	0	0	X	1

are added, updating the Boolean matrix as we see above.

- Updated `HotPaths`:

$$\texttt{HotPaths} = [\{\{e_1, e_2\}, \{2\}\}, \{\{e_2, e_3\}, \{3\}\}, \{\{e_3, e_5\}, \{3, 4\}\}]$$

- Updated Boolean Matrix:

$$\text{Boolean Matrix (Iteration 1)} = \begin{array}{c|cccc} & e_1 & e_2 & e_3 & e_5 \\ \hline e_1 & X & 1 & 0 & 0 \\ e_2 & 0 & X & 1 & 0 \\ e_3 & 0 & 0 & X & 1 \end{array}$$

updating now the Boolean Matrix for the next Iteration :

$$\text{Boolean Matrix (Iteration 1)} = \begin{array}{c|cccc} & e_1 & e_2 & e_3 & e_5 \\ \hline e_1, e_2 & 0 & X & 0 & 0 \\ e_2, e_3 & 0 & 0 & X & 0 \\ e_3, e_5 & 0 & 0 & 0 & X \end{array}$$

Since the HoMoEdge Detection Phase is complete, we mark all intersections in the Boolean matrix where the edge corresponding to the last edge of p (Horizontal Row: `p[len(p)]`) aligns with q (Vertical Column) as "X". This step prepares the Boolean matrix for the next iteration.

During the update process, we also evaluate the columns to identify and prune any edges that can no longer serve as potential connections from the last edge of any path (Horizontal Rows). In our example, we can prune the first column (e_1) and the second column (e_2) because these edges cannot be reached from the last edge of any path (Horizontal Rows).

$$\text{Boolean Matrix (Iteration 1)} = \begin{array}{c|cc} & e_3 & e_5 \\ \hline e_1, e_2 & 0 & 0 \\ e_2, e_3 & 0 & 0 \\ e_3, e_5 & X & 0 \end{array}$$

We can also prune the last line e_3, e_5 for the same reason as we did during the "Inputs and Initialization" phase at the beginning of our example.

$$\text{Boolean Matrix (Iteration 1)} = \begin{array}{c|cc} & e_3 & e_5 \\ \hline e_1, e_2 & 0 & 0 \\ e_2, e_3 & X & 0 \end{array}$$

Iteration : Detection of Hot Motion Paths, Length = 3

- Horizontal Rows (Current HoMoPaths of Length = 2): $\{\{e_1, e_2\}, \{e_2, e_3\}\}$.
- Vertical Columns (Remaining HoMoEdges): $\{e_3, e_5\}$.

- The algorithm evaluates connections. For example:

$$\widehat{J}(\{e_1, e_2\}, e_3) = \frac{|\varnothing|}{2} = 0 < 0.2$$

$$\text{Boolean Matrix (Iteration 2)} = \begin{array}{c|cc} & e_3 & e_5 \\ \hline e_1, e_2 & 0 & 0 \\ e_2, e_3 & X & 0 \end{array}$$

- Path $\{e_1, e_2, e_3\}$ is not added to `HotPaths`.

However:

$$\widehat{J}(\{e_2, e_3\}, e_5) = \frac{|\{3\}|}{2} = \frac{1}{2} > 0.2$$

$$\text{Boolean Matrix (Iteration 2)} = \begin{array}{c|cc} & e_3 & e_5 \\ \hline e_1, e_2 & 0 & 0 \\ e_2, e_3 & X & 1 \end{array}$$

- Path $\{e_2, e_3, e_5\}$ is added to `HotPaths`.

- Updated `HotPaths`:

$$\texttt{HotPaths} = [\{\{e_2, e_3, e_5\}, \{3\}\}]$$

- Updated Boolean Matrix for Next Iteration:

$$\text{Boolean Matrix (Iteration 2)} = \begin{array}{c|cc} & e_3 & e_5 \\ \hline e_2, e_3, e_5 & 0 & 0 \end{array}$$

Repeating the same updating/pruning process as already described during the flow , we prune columns e_3 and e_5 :

$$\text{Boolean Matrix (Iteration 2)} = \begin{array}{c|c} & \text{empty Horizontal Axe} \\ \hline e_2, e_3, e_5 & \end{array}$$

No further connections are possible.

Iteration : Termination

- The Boolean matrix is now empty, indicating that no further connections or extensions are possible. Therefore, the algorithm terminates.

- Final Output:

The final dictionary of Hot Motion Paths (`HoMoPaths_Dict`) is as follows:

$$\texttt{HoMoPaths\_Dict} = \begin{array}{l} \text{length:}\{\{\text{Edge/Path}\}, \{\text{Intersected MinHash Signature}\}\}, \\ 1 : \{\text{Empty (all HoMoEdges are part of longer HoMoPaths)}\}, \\ 2 : \{\{e_1, e_2\}, \{2\}\}, \\ 3 : \{\{e_2, e_3, e_5\}, \{3\}\}. \end{array}$$

As observed, the path $\{e_1, e_2\}$ is excluded from the dictionary for length = 2, since it is a subset of a longer path, $\{e_2, e_3, e_5\}$. Only maximal paths are retained in the final output to ensure efficiency and avoid redundancy.

5.4 MinHash Values and Size Estimation

Minhashing, principially as a powerful comparison tool, primarily abstracts away the absolute sizes and characteristics of datasets to focus solely on comparisons, and help us identify similar edges in order to construct hot motion paths. Beyond their typical use for similarity detection, Minhash signatures can also serve as a foundation for estimating the size of unions and intersections.

According to the previously introduced formula (4.3), to classify an edge as "hot," we require knowledge of both the size of the set of objects detected on each ($|O_{e_i}|$, and the total unique objects detected across all regions ($\left|\bigcup_{\forall R_g:\text{Intersects}(e_i,R_g)=\text{true}} O_{R_g}\right|$. In our application, we aim to estimate the size of object sets O_{e_i} observed on an edge e_i. Recall the formula for the hot motion support:

$$HoMoSup(e_i) = \frac{|O_{e_i}|}{\left|\bigcup_{\forall R_g:\text{Intersects}(e_i,R_g)=\text{true}} O_{R_g}\right|}.$$

To estimate the cardinalities efficiently, we leverage MinHash signatures. Each sensor associated with an edge e_i maintains a MinHash signature of its observed object set O_{e_i}.

There are two main tasks:

1. **Estimating the Size of Each Sensor's Own Object Set $|O_{e_i}|$:**

   For this, each sensor can choose between two methods:

   a) Extra Counter

   Each sensor sends to Regional Leader its minHash signature and an integer as the numeric count of observed unique real object IDs

   $$\{Counter_{e_l}, minHashSignature_{e_i}\}$$

b) (A-LESE) Atomic Least Element Size Estimation

Alternatively, each sensor sends only its minHash signature to Regional Leader

$$\{minHashSignature_{e_i}\}$$

and we estimate its real size, as :

$$\hat{n}_{e_i} = \frac{kH}{\sum_{j=1}^{k} m_j^{(e_i)}} - 1,$$

where $m_j^{(e_i)}$ are the MinHash minimum values for edge e_i, based on [20].

2. **Estimating the Size of the Union of Object Sets** $\left|\bigcup_{\forall R_g:\text{Intersects}(e_i,R_g)=\text{true}} O_{R_g}\right|$:

a) (G-LESE) Global Least Element Size Estimation:.

We compute:

$$\hat{n}_{\text{union}} = \frac{kH}{\sum_{j=1}^{k} m_j^{(\text{union})}} - 1, \tag{5.14}$$

where $m_j^{(\text{union})} = \min\left\{m_j^{(e_i)}, m_j^{(R_g)} \mid \text{Intersects}(e_i, R_g) = \text{true}\right\}$ are the combined MinHash minimum values across all intersecting regions, as derived by [20, 22].

b) (JISE) Jaccard Index Estimation Size:

We compute:

$$|U_i| = \frac{|U_{i-1}| + |A_i|}{1 + \hat{J}(U_{i-1}, A_i)}. \tag{5.15}$$

where $\hat{J}(U_{i-1}, A_i)$ is the estimated Jaccard similarity between the current union $U_{i-1} = A_1 \cup A_2 \cup \cdots \cup A_{i-1}$ and the next set A_i, derived from their respective MinHash signatures.

5.4.1 Set's Size Estimation : Extra Counter

In this approach, each sensor transmits its MinHash signature along with an integer representing the numeric count of observed unique object IDs:

$$\{Counter_{e_l}, MinHashSignature_{e_i}\}$$

The counter is incremented whenever the sensor detects a new unique object ID (tag ID), directly providing the cardinality $|O_{e_i}|$ without requiring additional estimations. This eliminates the need to estimate the atomic or local cardinality for each sensor, simplifying the process and reducing computational complexity.

Regional Leader Node($RegL_j$)	**Data Set** ($O_{e_{i..n}}$)
e_i	$\{Counter_{e_i}, minHashSignature_{e_i}\}$
$\vdots$	$\vdots$
e_h	$\{Counter_{e_h}, minHashSignature_{e_h}\}$
$\vdots$	$\vdots$
e_k	$\{Counter_{e_k}, minHashSignature_{e_k}\}$
$\vdots$	$\vdots$
e_l	$\{Counter_{e_l}, minHashSignature_{e_l}\}$
$\vdots$	$\vdots$
e_m	$\{Counter_{e_m}, minHashSignature_{e_m}\}$
$\vdots$	$\vdots$
e_n	$\{Counter_{e_n}, minHashSignature_{e_n}\}$

Table 5.2: Exemplary Data Organization at the Regional Leader Tier Using MinHash Signatures

Including this integer in the transmission payload introduces negligible communication overhead, as it is a single value. However, it significantly enhances the efficiency and accuracy of downstream operations, such as determining the overlap and intersections between sets. This straightforward addition ensures that payloads remain lightweight while improving the precision of data aggregation at the Regional Leader level.

5.4.2 Set's Size Estimation: Atomic Least Element Size Estimation (A-LESE)

Let S be a finite set with an unknown cardinality $n = |S|$. To estimate n, we use MinHash signatures derived from random hash functions. MinHash efficiently estimates cardinality by leveraging random permutations and the distribution of minimum hash values, following Cohen's Size-Estimation Framework [20]. We present the theoretical foundation established by Cohen.

MinHash Signatures and Random Hash Functions

Consider a family of k independent hash functions $\{h_i\}_{i=1}^{k}$, where each $h_i : S \to [0, H]$ maps elements of S to real numbers uniformly at random over the interval $[0, H]$, with $H > 0$ (commonly $H = 1$ for simplicity). Each h_i effectively simulates a random hash function of the elements in S.

For each hash function h_i, compute the minimum hash value over S:

$$m_i = \min_{x \in S} h_i(x).$$

where $m_i =$ minimum hash value under h_i, $x =$ an element of S, $S =$ the finite set.

Distribution of Minimum Hash Values

Understanding the distribution of the minimum hash values m_i is crucial for deriving the cardinality estimator. The cumulative distribution function (CDF) of m_i is:

$$F_m(t) = 1 - \left(1 - \frac{t}{H}\right)^n, \quad 0 \le t \le H.$$

where $F_m(t) = \Pr[m_i \le t]$, $H =$ upper bound on the hash range, $n = |S|$.

Proof. Each hash value $h_i(x)$ is independently and uniformly distributed over $[0, H]$. The probability that a single hash value exceeds t is:

$$\Pr[h_i(x) > t] = 1 - \frac{t}{H}.$$

Since there are n elements in S, the probability that *all* n hash values exceed t is:

$$\Pr[m_i > t] = \left(1 - \frac{t}{H}\right)^n.$$

Hence,

$$F_m(t) = 1 - \Pr[m_i > t] = 1 - \left(1 - \frac{t}{H}\right)^n.$$

□

Expected Value of the Minimum Hash Value

The expected value of m_i is:

$$\mathbb{E}[m_i] = \frac{H}{n+1}., \text{ where } H = \text{hash range upper bound}, n = |S|.$$

Proof. First, recall the CDF of m_i:

$$F_m(t) = 1 - \left(1 - \frac{t}{H}\right)^n.$$

Differentiating $F_m(t)$ with respect to t gives the probability density function (PDF) [34]:

$$f_m(t) = \frac{d}{dt}F_m(t) = \frac{n}{H}\left(1 - \frac{t}{H}\right)^{n-1}.$$

The expected value of m_i is then:

$$\mathbb{E}[m_i] = \int_0^H t\, f_m(t)\, dt = \int_0^H t \cdot \frac{n}{H}\left(1 - \frac{t}{H}\right)^{n-1} dt.$$

Define the substitution $u = \frac{t}{H}$. Then $t = Hu$ and $dt = H\, du$. The integral becomes:

$$\mathbb{E}[m_i] = \frac{n}{H}\int_0^H t\left(1 - \frac{t}{H}\right)^{n-1} dt = \frac{n}{H}\int_0^1 \Big(Hu\Big)\,(1-u)^{n-1}\,\Big(H\, du\Big).$$

$$= nH\int_0^1 u\,(1-u)^{n-1} du.$$

Using the Beta function relationship [32], :

$$\int_0^1 u^{\alpha-1}(1-u)^{\beta-1} du = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)},$$

we set $\alpha = 2$, $\beta = n$. Then (reference to Johnson, Kotz, Balakrishnan [33]) :

$$\int_0^1 u(1-u)^{n-1} du = \frac{\Gamma(2)\,\Gamma(n)}{\Gamma(n+2)} = \frac{1!\,(n-1)!}{(n+1)!} = \frac{1}{n(n+1)}.$$

Thus,

$$\mathbb{E}[m_i] = nH \cdot \frac{1}{n(n+1)} = \frac{H}{n+1}.$$

□

Derivation of the Cardinality Estimation Formula

Using the result from Section 5.4.2, we have:

$$\mathbb{E}[m_i] = \frac{H}{n+1}. \tag{5.16}$$

where $\mathbb{E}[m_i]$ = expected minimum hash value, H = hash range upper bound, $n = |S|$.

Solving for n:

$$n = \frac{H}{\mathbb{E}[m_i]} - 1.$$

In practice, $\mathbb{E}[m_i]$ is replaced by the empirical average $\overline{m}$ of the observed $\{m_i\}_{i=1}^{k}$:

$$\overline{m} = \frac{1}{k}\sum_{i=1}^{k} m_i. \tag{5.17}$$

where k = number of hash functions, m_i = minimum hash value under the i-th hash function.

Thus, the estimator for n becomes:

$$\hat{n} = \frac{H}{\overline{m}} - 1 = \frac{H}{\left(\frac{1}{k}\sum_{i=1}^{k} m_i\right)} - 1 = \frac{kH}{\sum_{i=1}^{k} m_i} - 1. \tag{5.18}$$

Normalization of the Hash Value Range

In the estimator (5.18), the MinHash values m_i lie within the range $[0, H]$. If the hash functions generate MinHash values with q bits, this implies a range of $[0, 2^q - 1]$. To normalize this range to $[0, 1]$, we express the estimator in the equivalent form:

$$\frac{k}{\left(\frac{1}{H}\sum_{i=1}^{k} m_i\right)} \overset{\text{(with range (0,1])}}{=} \frac{k}{\sum_{i=1}^{k} m_{\text{normalize},i}} \overset{\text{(with range [0,1])}}{=} \left(\frac{1}{k}\sum_{i=1}^{k} m_{\text{normalized},i}\right)^{-1} \tag{5.19}$$

Since each $m_i \in (0, H]$ maps to $m_i/H \in [0, 1]$, this normalization standardizes MinHash values while maintaining estimation integrity.

Algorithm 2 A-LESE : Atomic Least Element Size Estimation - for a Set

Input: $A_1, A_2, \ldots, A_n$, k, H
Output: $\{\hat{n}_1, \ldots, \hat{n}_n\}$
function ComputeMinHash(A, k, H)
    Initialize $M \leftarrow [H] \times k$
    for each $x \in A$ **do**
        for $j = 1$ to k **do**
            $M[j] \leftarrow \min(M[j], h_j(x))$
        end for
    end for
    return M
end function
function EstimateCardinality(M, k, H)
    $\hat{n} \leftarrow \frac{k \times H}{\sum M} - 1$
    return $\hat{n}$
end function
for $i = 1$ to n **do**
    $M_{A_i} \leftarrow$ ComputeMinHash(A_i, k, H)
    $\hat{n}_i \leftarrow$ EstimateCardinality(M_{A_i}, k, H)
end for
Return $\{\hat{n}_1, \ldots, \hat{n}_n\}$

A-LESE Flowchart :

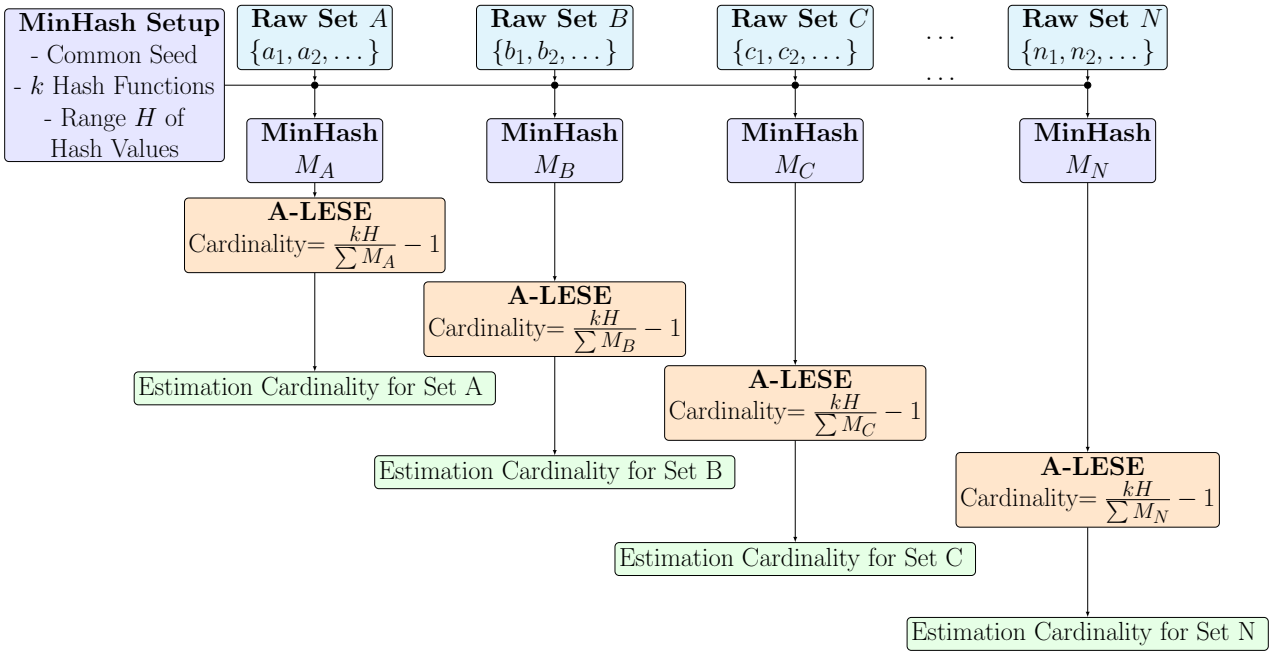


Figure 5.2: Process of the A-LESE algorithm.

5.4.3 Global Size Estimation : Global Least Element Size Estimation (G-LESE)

Cohen's Size-Estimation Framework [20] introduced a foundational approach for estimating set cardinalities in large data environments. This framework, originally designed to compute the size of individual sets, was later extended to address the more complex problem of estimating the cardinality of the union of multiple sets $S_1, S_2, \ldots, S_r$ [21]. The Global Least Element Size Estimation (G-LESE) method leverages this extension to efficiently approximate the size of unions without explicitly computing them, thereby minimizing computational and memory overhead.

Consider the union of sets $S = \bigcup_{j=1}^{r} S_j$ with $n = |S|$ representing the cardinality of the union. Direct computation of n through enumeration of S becomes infeasible as the size of S_j grows or as the number of sets increases. Let $h_1, h_2, \ldots, h_k$ denote k independent hash functions, each mapping elements to a range $[0, H]$, where H represents the upper bound of the hash values.

For each set S_j, the algorithm computes a vector of minimum hash values:

$$M_j[i] = \min_{x \in S_j} h_i(x), \quad \text{for } i = 1, \ldots, k.$$

This results in a compact representation $M_j = [M_j[1], M_j[2], \ldots, M_j[k]]$ that characterizes the set S_j. The overall estimate of the union is derived by taking the element-wise minimum of these vectors:

$$M_{\text{union}}[i] = \min(m_1[i], m_2[i], \ldots, m_r[i]), \quad \text{for } i = 1, \ldots, k.$$

The final estimation of the cardinality $\hat{n}$ of the union S is given by

$$\hat{n} = \frac{kH}{\sum_{i=1}^{k} M_{\text{union}}[i]} - 1. \tag{5.20}$$

Here, H represents the largest possible hash value, ensuring that the estimate scales appropriately with the hashed space. The sum of the minimum hash values $\sum M_{\text{union}}$ reflects the sparsity of the hashed elements, allowing the algorithm to approximate the cardinality without explicit enumeration. The algorithm's efficiency is enhanced by the fact that each set is processed independently, allowing parallel execution across distributed systems.

Recall from 5.19 , we are allowed to normalize the minimum hash values of the $\sum M_{\text{union}}$ in the range $m_i/H \in [0, 1]$, having the equivalent :

$$\hat{n} = \frac{kH}{\sum_{i=1}^{k} M_{\text{union}}[i]} - 1 \overset{\text{(with range (0,1])}}{=} \frac{k}{\sum_{i=1}^{k} \frac{M_{\text{union}}[i]}{H}} - 1 \overset{\text{(with range [0,1])}}{=} \left(\frac{1}{k} \sum_{i=1}^{k} M_{\text{normalized},i} \right)^{-1} - 1. \tag{5.21}$$

Algorithm 3 G-LESE : Global Least Element Size Estimation - for Union of Sets

Input: $A_1, A_2, \ldots, A_n$, k, H
Output: $\hat{n}_{\text{union}}$
function ComputeMinHash(A, k, H)
    Initialize $M \leftarrow [H] \times k$
    for each $x \in A$ **do**
        for $j = 1$ to k **do**
            $M[j] \leftarrow \min(M[j], h_j(x))$
        end for
    end for
    return M
end function
function EstimateCardinality(M, k, H)
    $\hat{n} \leftarrow \frac{k \times H}{\sum M} - 1$
    return $\hat{n}$
end function
for $i = 1$ to n **do**
    $M_{A_i} \leftarrow$ ComputeMinHash(A_i, k, H)
end for
$M_{\text{union}} \leftarrow \min(M_{A_1}, M_{A_2}, \ldots, M_{A_n})$
$\hat{n}_{\text{union}} \leftarrow$ EstimateCardinality(M_{union}, k, H)
Return $\hat{n}_{\text{union}}$

G-LESE Flowchart :

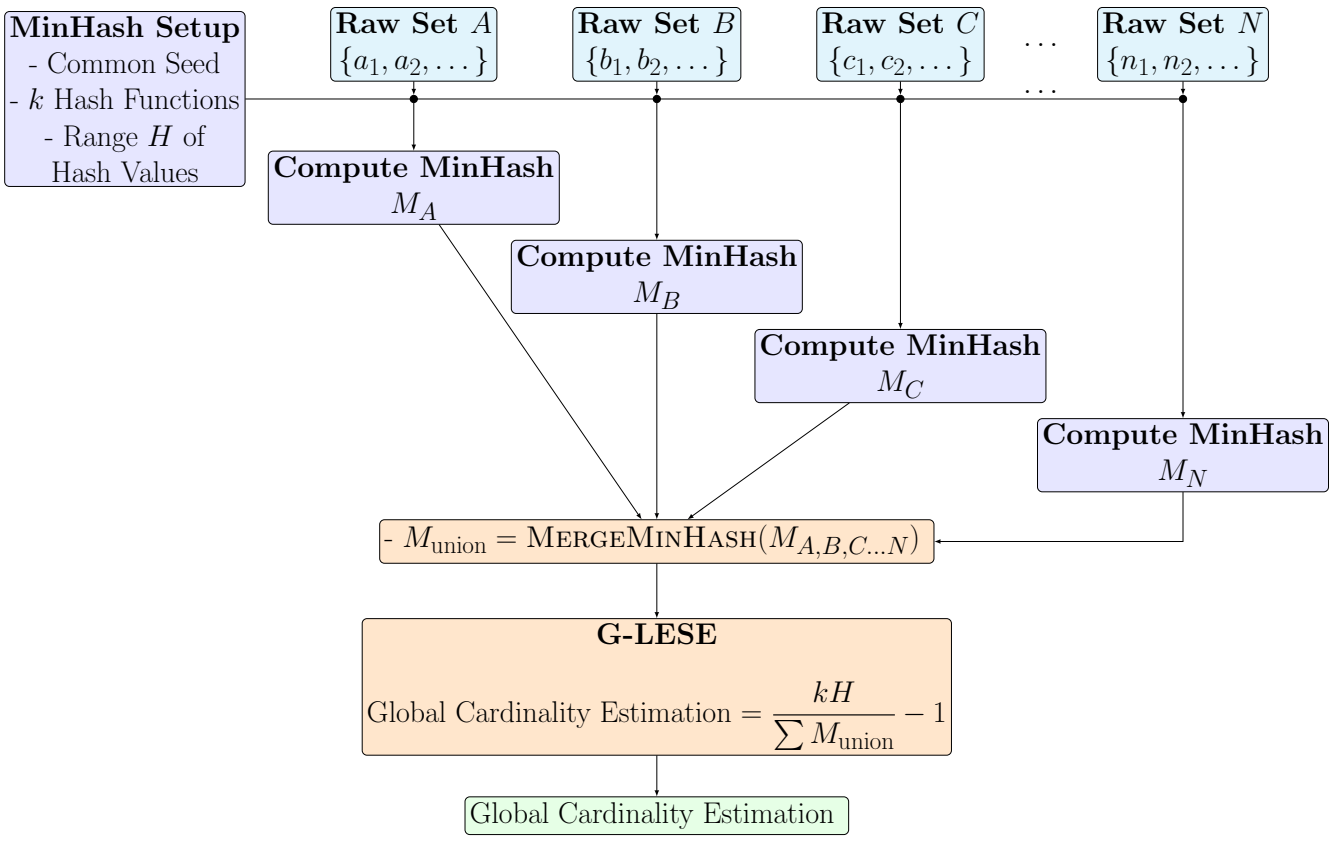


Figure 5.3: Process of the G-LESE algorithm.

5.4.4 High-Probability Bounds for A-LESE and G-LESE via Hoeffding's Inequality

As established by Cohen [20, 22] , the "least-element" estimator for set size is *unbiased* and achieves a high-probability relative-error guarantee with $k = O\big(\frac{1}{\varepsilon^2}\ln\frac{1}{\delta}\big)$ hash functions, using direct variance and Chebyshev-type bounds. Here, we give an equivalent high-probability error bound by applying *Hoeffding's inequality* to the i.i.d. MinHash values.

Setup. Consider a set of size n, and let each (normalized) MinHash value be denoted by

$$m_{\text{normalized},i} \;\in\; [0,1], \quad i = 1,\ldots,k.$$

Since the MinHash functions are uniform, from prior equations (see Eqs. (2.1) and (2.2) in [20] for instance), we have

$$\overline{m}_{\text{normalized}} \;=\; \frac{1}{k}\sum_{i=1}^{k} m_{\text{normalized},i}, \qquad \mathbb{E}[\overline{m}_{\text{normalized}}] \;=\; \frac{1}{n+1}. \tag{5.22}$$

The *size estimator* (cf. [20]) is then

$$\hat{n} \;=\; \big(\overline{m}_{\text{normalized}}\big)^{-1} \;-\; 1.$$

Applying Hoeffding's Inequality to $\overline{m}$

If $X_1,\ldots,X_k$ are i.i.d. random variables, each bounded by $[0,1]$ with mean μ, then

$$\Pr\Big(\,\big|\tfrac{1}{k}\sum_{i=1}^{k} X_i - \mu\big| \;\geq\; t\Big) \;\leq\; 2\,\exp\big(-2\,k\,t^2\big).$$

In our setting, $X_i = m_{\text{normalized},i} \in [0,1]$, and $\mu = \frac{1}{n+1}$. Thus,

$$\Pr\Big[\,\big|\overline{m} - \mathbb{E}[\overline{m}]\big| \;\geq\; t\Big] \;\leq\; 2\,\exp(-2\,k\,t^2).$$

$$\Pr\Big[\,\big|\frac{1}{k}\sum_{i=1}^{k} m_{\text{normalized},i} - \tfrac{1}{n+1}\big| \;\geq\; t\Big] \;\leq\; 2\,\exp(-2\,k\,t^2).$$

We often care about a relative deviation of $\overline{m}$ from its mean, e.g. letting $t = \frac{\varepsilon}{n+1}$ gives

$$\Pr\Big[\,\big|\overline{m}_{normalized} - \tfrac{1}{n+1}\big| \;\geq\; \tfrac{\varepsilon}{n+1}\Big] \;\leq\; 2\,\exp(-2\,k\,\varepsilon^2).$$

The exponent in Hoeffding's bound remains in terms of ε^2 rather than incorporating an additional $(n+1)^2$ factor. Hoeffding's inequality inherently controls deviations in the empirical mean, and our transformation $t = \varepsilon/(n+1)$ is already reflected in the bound ,confirmed in prior works [29, 30, 31].

Bounding the Error in $\hat{n}$

To derive an error bound for our estimator $\hat{n}$, we must first understand how deviations in $\overline{m}$ propagate to deviations in $\hat{n}$. Since our estimator is defined as: $\hat{n} = \overline{m}^{-1} - 1$, we observe:

$$|\hat{n} - n| \;=\; \left|\overline{m}^{-1} - 1 \;-\; n\right| \;=\; \left|\overline{m}^{-1} - (n+1)\right|. \tag{5.23}$$

Given that $\overline{m}$ is an empirical mean that fluctuates around its expected value (5.22), we can use a first-order Taylor expansion around $m_0 = \frac{1}{n+1}$ for the function $f(m) = m^{-1}$.

Taylor's theorem provides a way to approximate a function locally when the function is differentiable. Taylor's approximation is valid because:
1. The function $f(m) = 1/m$ is differentiable for all $m > 0$.
2. The empirical mean $\overline{m}$ is **highly concentrated around** m_0 due to Hoeffding's bound, meaning its deviations are small.

Thus, the first-order expansion provides a precise and controlled approximation. Since our function is:

$$f(m) = m^{-1} \quad , \quad \text{and its derivative is} \quad f'(m) = -m^{-2},$$

Evaluating at m_0:

$$f'(m_0) = -\left(\frac{1}{n+1}\right)^{-2} = -(n+1)^2.$$

Using the first-order Taylor approximation:

$$\overline{m}^{-1} = f(m_0) + f'(m_0)(\overline{m} - m_0).$$

Since $f(m_0) = n + 1$, we obtain:

$$\overline{m}^{-1} = (n+1) + (n+1)^2(\overline{m} - m_0). \tag{5.24}$$

Connecting Hoeffding's Bound to $\hat{n}$

Using the Taylor approximation, and based on 5.24, we rewrite the equation 5.23:

$$|\hat{n} - n| = \left|\overline{m}^{-1} - (n+1)\right|$$

as :

$$|\hat{n} - n| \;=\; \left|(n+1) + (n+1)^2(\overline{m} - m_0) - (n+1)\right| \;=\; (n+1)^2\left|\overline{m} - \frac{1}{n+1}\right|. \tag{5.25}$$

From Hoeffding's inequality, the empirical mean satisfies:

$$P\left(|\overline{m} - \mathbb{E}[\overline{m}]| \geq \frac{\varepsilon}{n+1}\right) \leq 2\exp(-2k\varepsilon^2).$$

Hoeffding's inequality Relative Error Bounding

Instead of bounding $|\hat{n} - n|$, we consider the relative error $\frac{|\hat{n}-n|}{n+1} \leq \varepsilon$. Using relative error ensures that our error bound *scales appropriately with the set size* n. If we used absolute error, the bound would be constant across different values of n, meaning the accuracy requirement would be disproportionately strict for small sets and too lenient for large sets. Relative error keeps the estimation accuracy meaningful across different set sizes.

Step 1: Transforming the Absolute Error Bound. We previously derived:

$$|\hat{n} - n| = (n+1)^2 |\overline{m} - \mathbb{E}[\overline{m}]|.$$

To transition to a relative error bound, we divide both sides by $n + 1$:

$$\frac{|\hat{n} - n|}{n+1} = (n+1)|\overline{m} - \mathbb{E}[\overline{m}]|.$$

Step 2: Applying Hoeffding's Inequality. From Hoeffding's inequality, the empirical mean satisfies:

$$P\left(|\overline{m} - \mathbb{E}[\overline{m}]| \geq \frac{\varepsilon}{n+1}\right) \leq 2\exp(-2k\varepsilon^2).$$

Multiplying both sides of the inequality by $(n + 1)$, we get:

$$P\left((n+1)|\overline{m} - \mathbb{E}[\overline{m}]| \geq \varepsilon\right) \leq 2\exp(-2k\varepsilon^2).$$

Since we showed that:

$$\frac{|\hat{n} - n|}{n+1} = (n+1)|\overline{m} - \mathbb{E}[\overline{m}]|,$$

this directly gives:

$$P\left(\frac{|\hat{n} - n|}{n+1} \geq \varepsilon\right) \leq 2\exp(-2k\varepsilon^2).$$

This result shows that instead of bounding the absolute deviation $|\hat{n} - n|$, we control the probability that the **relative error** exceeds ε. Since the probability bound remains unchanged, the error bound is now valid across all values of n in a dimensionally consistent way.

Setting the right side of this inequality to δ gives us:

$$\delta = 2e^{-2k\epsilon^2}. \tag{5.26}$$

Solving for k gives the number of hash functions needed to ensure. Hence, to ensure , for a set(A-LESE) or the union of sets(G-LESE) , $\pm\varepsilon$ relative error (with probability at least $1 - \delta$), we require

$$k = O\left(\frac{\ln(2/\delta)}{2\epsilon^2}\right). \tag{5.27}$$

This matches the well-known asymptotic bounds from Cohen's original analysis [20] (which used Chebyshev-type variance arguments).

High-Probability Tail Bound via Hoeffding's Inequality

We provide some numeric results regarding the number of hash functions predicted by two different bounding strategies for Cohen's size estimator: a *naive CLT-based bound* versus our *Hoeffding-based* approach.

Error (%)	**Method**	**Precision/Confidence Level (%)**							
		70	**75**	**80**	**85**	**90**	**95**	**99**	**99.9**
25.0	CLT	20	24	29	36	46	64	109	176
	Hoeffding	16	17	19	21	24	30	43	61
	Improvement	1.25	1.41	1.53	1.71	1.92	2.13	2.53	2.89
15.0	CLT	51	61	75	95	123	173	297	484
	Hoeffding	43	47	52	58	67	82	118	169
	Improvement	1.19	1.30	1.44	1.64	1.84	2.11	2.52	2.86
5.0	CLT	435	531	658	832	1085	1539	2657	4335
	Hoeffding	380	416	461	519	600	738	1060	1521
	Improvement	1.14	1.28	1.43	1.60	1.81	2.09	2.51	2.85
1.0	CLT	10818	13227	16386	20738	27063	38418	66360	108309
	Hoeffding	9486	10398	11513	12952	14979	18445	26492	38005
	Improvement	1.14	1.27	1.42	1.60	1.81	2.08	2.50	2.85

Table 5.3: Comparison of Cohen's and Hoeffding's Sample Size Bounds

Cohen's *unbiasedness* results for A-LESE and G-LESE remain fully valid. The high-probability $\pm\varepsilon$ error bound itself was already established in [20] via variance-based arguments; here, we simply re-derive it using a first-order Taylor expansion and Hoeffding's inequality. By leveraging the boundedness of the normalized MinHash variables, we use a different route to confirm the already known bound from Cohen. Our bounding approach is somewhat different ,and perhaps, simpler by modern standards, providing a more direct modern derivation for completeness and clarity. This yields the same asymptotic scaling but offers a succinct modern derivation

5.4.5 Global Size Estimation : Jaccard Index Size Estimation - JISE

For two sets:

For two sets A and B with known cardinalities $|A|$ and $|B|$, and a Jaccard Similarity $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$ or based on equation 5.7, the estimated Jaccard Similarity $\hat{J}(A,B) = \frac{m}{k}$ derived from their MinHash signatures, the union size $|A \cup B|$ is estimated using the formula:

$$|A \cup B| = \frac{|A| + |B|}{1 + J(A,B)} \sim \frac{|A| + |B|}{1 + \hat{J}(A,B)}. \tag{5.28}$$

This formula is derived from the definition of Jaccard similarity and the inclusion-exclusion principle:

$$\begin{aligned} J(A,B) &= \frac{|A \cap B|}{|A \cup B|} \\ \Leftrightarrow |A \cup B| &= |A| + |B| - |A \cap B| \\ \Leftrightarrow |A \cup B| &= \frac{|A| + |B|}{1 + J(A,B)} \end{aligned} \tag{5.29}$$

For multiple sets

We want to extend the previous idea and use it for multiple sets $A_1, A_2, \ldots, A_n$, the algorithm iteratively applies the union size estimation. Starting with $U_1 = A_1$, for each subsequent set A_i $(i = 2, 3, \ldots, n)$, the union U_i is estimated as:

$$|U_i| = \frac{|U_{i-1}| + |A_i|}{1 + J(U_{i-1}, A_i)} \sim \frac{|U_{i-1}| + |A_i|}{1 + \hat{J}(U_{i-1}, A_i)}. \tag{5.30}$$

Here, $\hat{J}(U_{i-1}, A_i)$ is the estimated Jaccard similarity between the current union $U_{i-1} = A_1 \cup A_2 \cup \cdots \cup A_{i-1}$ and the next set A_i, derived from their respective MinHash signatures. At each iteration, a new MinHash signature must be generated for the union U_i. This is achieved by merging the MinHash signatures of the previously estimated union U_{i-1} and the next set A_i. The resulting signature represents the combination:

$$M_U = \textsc{MergeMinHash}(M_{U_{i-1}}, M_{A_i}).$$

This ensures that subsequent comparisons between U_i and A_{i+1} are based on an accurate estimate of the intersection size. Without regenerating the MinHash signature for the union at each step, the estimator

$$\hat{J}(U_i, A_{i+1})$$

may fail to reflect the true overlap, compromising the accuracy of the overall size estimation.

Theoretical Verification of the Proposed Algorithm JISE

Global Accuracy

The JISE algorithm builds upon the foundational principles of the Jaccard Index and MinHashing, extending them to operate effectively on more than just pairs of sets. While the Jaccard Index and MinHashing principles provide a well-defined mechanism for measuring similarity and bounding errors between two sets (as described in Section 5.3), JISE introduces a sequential approach. This extension enables the algorithm to capture and evaluate the global overlap among multiple sets, rather than limiting the analysis to pairwise comparisons.

The error in $|U_i|$ is due to the difference between $\hat{J}(U_{i-1}, A_i)$ and $J(U_{i-1}, A_i)$. Define:

$$D_i = 1 + \hat{J}(U_{i-1}, A_i), \quad D_i^{\text{true}} = 1 + J(U_{i-1}, A_i.$$

The error is:

$$|U_i| - |U_i|_{\text{true}} = C_i \cdot \left(\frac{1}{D_i} - \frac{1}{D_i^{\text{true}}} \right),$$

where $C_i = |U_{i-1}| + |A_i|$.

Expanding $\frac{1}{D_i} - \frac{1}{D_i^{\text{true}}}$ yields:

$$\frac{1}{D_i} - \frac{1}{D_i^{\text{true}}} = \frac{D_i^{\text{true}} - D_i}{D_i \cdot D_i^{\text{true}}}.$$

Substituting $D_i = 1 + \hat{J}(U_{i-1}, A_i)$ and $D_i^{\text{true}} = 1 + J(U_{i-1}, A_i)$, we have:

$$|U_i| - |U_i|_{\text{true}} = C_i \cdot \frac{J(U_{i-1}, A_i) - \hat{J}(U_{i-1}, A_i)}{D_i \cdot D_i^{\text{true}}}.$$

Bounding the Error

Using Hoeffding's inequality [26], the probability that the estimated Jaccard similarity deviates from the true value is:

$$P\left(|\hat{J}(U_{i-1}, A_i) - J(U_{i-1}, A_i)| \geq \epsilon \right) \leq 2\exp(-2k\epsilon^2).$$

To ensure this bound holds with confidence $1 - \delta$, we set: $2\exp(-2k\epsilon^2) \leq \delta_i$.

$$\iff -2k\epsilon^2 \leq \ln\left(\frac{\delta_i}{2}\right) \iff k \geq \frac{\ln(2/\delta_i)}{2\epsilon^2}.$$

Global Confidence and Propagation of Errors

To estimate the union size $|U_n|$ across n sets, the total confidence must remain $1 - \delta$. Using the union bound:

$$P\left(\bigcup_{i=1}^{n} |\hat{J}(U_{i-1}, A_i) - J(U_{i-1}, A_i)| \geq \epsilon\right) \leq \sum_{i=1}^{n} \delta_i.$$

Setting $\delta_i = \frac{\delta}{n}$, the total confidence is:

$$\sum_{i=1}^{n} \delta_i = \delta.$$

Substituting $\delta_i = \frac{\delta}{n}$ into the bound for k, we provide the final bound as:

$$k = O\left(\frac{\ln(2n/\delta)}{2\epsilon^2}\right). \tag{5.31}$$

Time Complexity Analysis

Understanding the computational efficiency of the algorithm is essential for its applicability to large-scale datasets. The time complexity analysis considers the number of sets n and the number of hash functions k.

Estimating $J(U_{i-1}, A_i)$ involves comparing k hash values, resulting in $\mathcal{O}(k)$ time per estimation. Across $n - 1$ iterations:

$$\mathcal{O}\left(k \cdot (n-1)\right)$$

$$\mathcal{O}(1) \quad \text{per iteration}$$

The arithmetic operations to compute $|U_i|$ are constant time. The time complexity per run is:

$$\mathcal{O}\left(k \cdot n - k\right)$$

Since n is significantly greater than 1 ($n \gg 1$) and the term $k \cdot n$ dominates k, the overall computational complexity can be expressed as:

$$\mathcal{O}(k \cdot n),$$

This means that the algorithm exhibits linear time complexity with respect to both the number of hash functions k and the number of sets n. This linear scalability ensures that the algorithm remains computationally efficient even as the dataset grows, provided that k is chosen carefully to balance the trade-off between accuracy and computational resources.

Thus, we introduce the JISE Algorithm as a Count-Distinct solution specifically designed to operate across multiple sets:

Algorithm 4 JISE : Jaccard Index Size Estimation - for Union of Sets

Input: $A_1, A_2, \ldots, A_n$, k, H
Output: $|U_n|$
$U \leftarrow A_1$
$M_U \leftarrow \textsc{ComputeMinHash}(U, k, H)$
for $i = 2$ to n **do**
    $M_{A_i} \leftarrow \textsc{ComputeMinHash}(A_i, k, H)$
    $\hat{J} \leftarrow \textsc{EstimateJaccard}(M_U, M_{A_i})$
    $|U| \leftarrow \frac{|U|+|A_i|}{1+\hat{J}}$
    $M_U \leftarrow \textsc{MergeMinHash}(M_U, M_{A_i})$
end for
Return $|U|$

JISE Flowchart :

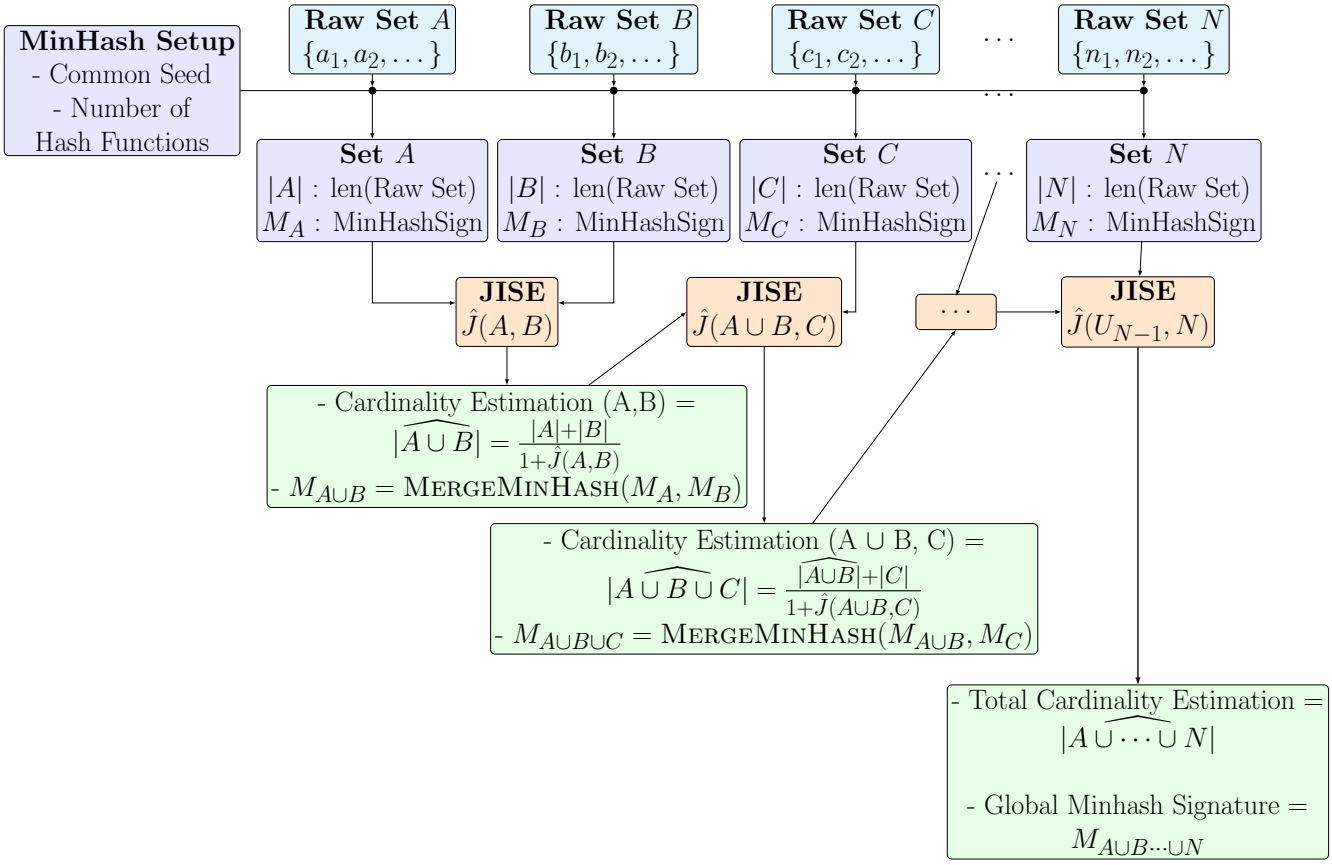


Figure 5.4: Process of algorithm JISE.

5.5 Comparison of G-LESE and JISE Algorithms

In this section, we present the experimental results obtained from comparing the G-LESE (Global Least Element Size Estimation) and JISE (Jaccard Index Size Estimation) algorithms. We focus on the *Absolute Relative Error* (ARE). Our goal is to determine under which circumstances each algorithm performs better, particularly in relation to the number of hash functions used (*NumHashFunc*), the overlap level among the sets (*Overlap*), and the sizes and number of the sets.

5.5.1 Error Metric

Before delving into the results, we define the error metric used to evaluate the algorithms:

- **Absolute Relative Error**: This metric measures the average of the absolute relative(percentage) errors between the estimated and actual union sizes , defined as:

$$\text{MAPE} = \left(\frac{1}{n} \sum_{i=1}^{n} \left| \frac{\text{Estimated}_i - \text{Actual}_i}{\text{Actual}_i} \right| \right) \times 100\%$$

5.5.2 Analysis of JISE and G-LESE Under Varying Parameters

Set-up :

We generated sets with overlap values of 0.2, 0.5, and 0.8. For each overlap value, more than 10,000 random experiments were conducted. Each experiment was repeated once using the same seed and three times with different seeds to compute an average. The plots below illustrate the average error rate of a single experiment after three repetitions with three distinct seeds. Additionally, the plots illustrate the degree of compression achieved for specific numbers of hash functions, which are annotated along the x-axis. The corresponding number of hash functions is also provided in the legend under the notation "HashFunc".

In the plots, we also provide the error values for both JISE and G-LESE for each case, presented as small notations above the line graph in the format (Error JISE, Error G-LESE) , in order to provide comprehensive and readable plots. To summarize the results, we present the following:

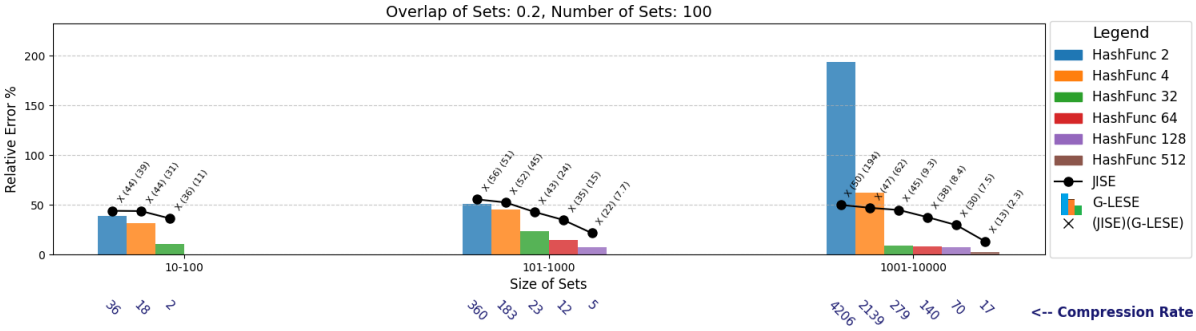


Figure 5.5: JISE vs G-LESE , Overlap = 0.2 , Number of Sets = 100

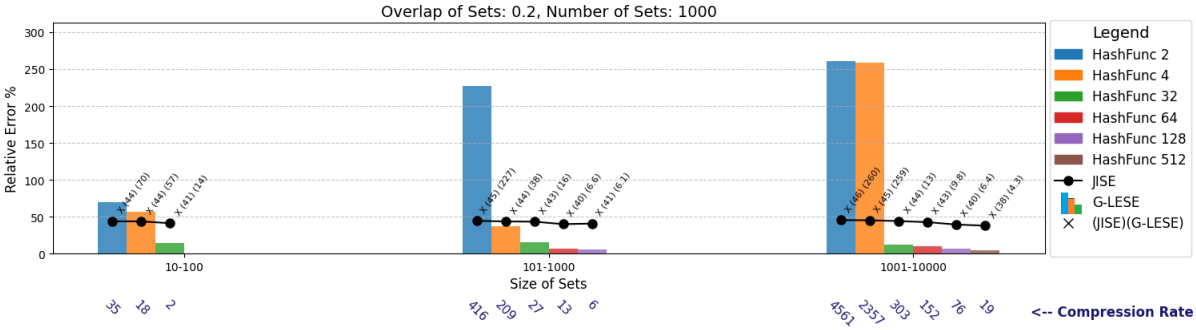


Figure 5.6: JISE vs G-LESE , Overlap = 0.2 , Number of Sets = 1000

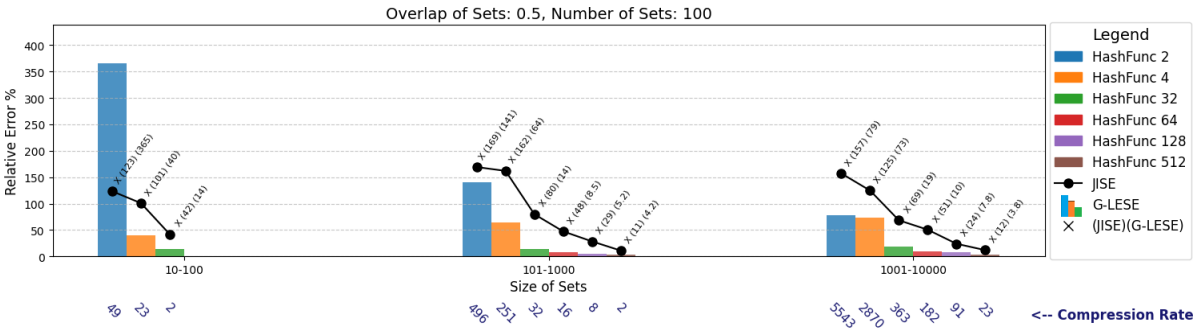


Figure 5.7: JISE vs G-LESE , Overlap = 0.5 , Number of Sets = 100

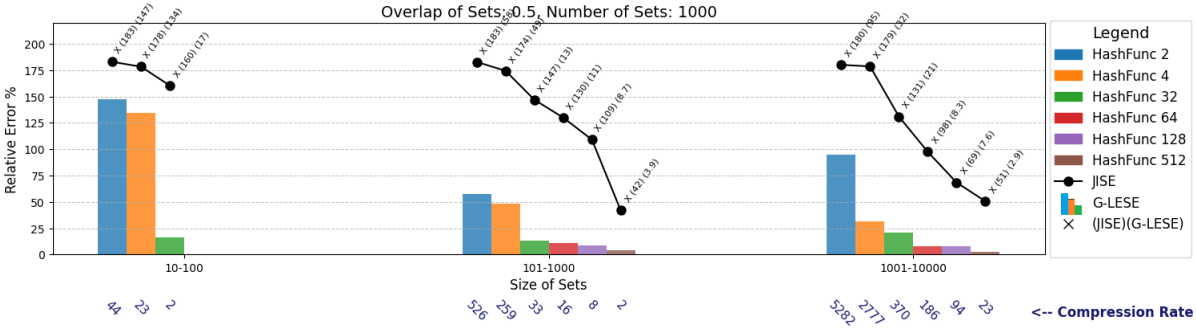


Figure 5.8: JISE vs G-LESE , Overlap = 0.5 , Number of Sets = 1000

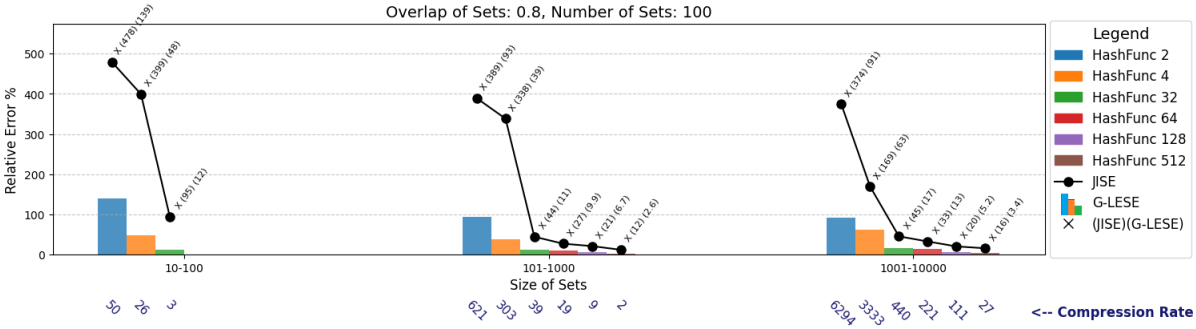


Figure 5.9: JISE vs G-LESE , Overlap = 0.8 , Number of Sets = 100

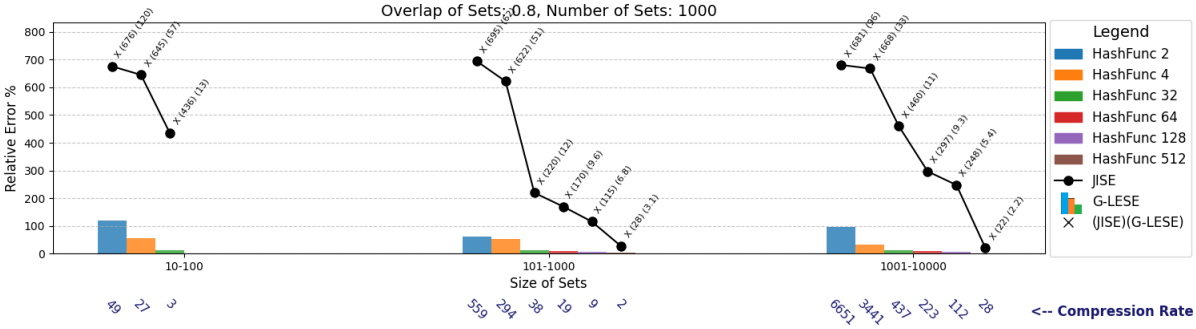


Figure 5.10: JISE vs G-LESE , Overlap = 0.8 , Number of Sets = 1000

JISE:

JISE's performance is highly sensitive to the number of hash functions and the degree of overlap between sets:

- **Low number of Hash Functios (2 to 4)** JISE performs decent but still worse than G-LESE for lower number of hash functions, when dealing with small set sizes (e.g., $|A|, |B| \approx 100$) or low-overlap datasets (overlap ≈ 0.2). Under these conditions, JISE demonstrates higher accuracy due to minimal set intersection, where hash functions sufficiently capture the differences between sets.

- **Higher number of Hash Functions (≥ 32):** As the number of hash functions increases (e.g., 32 or 64), JISE stabilizes, delivering more reliable performance across varying overlap levels, set sizes, and quantities. This highlights the importance of higher hash-functions counts for achieving consistent results.

- **High Overlap (≈ 0.8):** In datasets with substantial overlap, JISE's accuracy declines notably under low amount of hash functions conditions, leading to incorrect quality estimations. However, performance improves considerably with higher number of hash functions, emphasizing JISE's dependence on extensive computations to counteract errors in high-overlap scenarios.

G-LESE:

G-LESE demonstrates robust and consistent performance across diverse dataset conditions, displaying minimal sensitivity to overlap, set size, or hash-functions count:

- **Low number of Hash Function:** With a minimal number of hash functions (e.g., 2 to 4), G-LESE performs slightly better than JISE, and makes it more suitable in scenarios with limited computational resources, offering slightly better accuracy and reliability.
- **Higher number of Hash Functions:** As the number of hash functions increases, G-LESE scales effectively, exhibiting great performance improvements, outperforming consistently JISE, under all different scenarios. This makes G-LESE particularly advantageous for large-scale datasets, ensuring computational efficiency without compromising accuracy.
- **Overlap and Set Size:** Unlike JISE, G-LESE's performance remains unaffected by increases in overlap or set size. It consistently delivers superior results across all overlap levels and dataset sizes, reinforcing its reliability across a wide range of scenarios.

Why G-LESE is Preferred Over JISE

G-LESE consistently outperforms JISE across all relevant factors—overlap, set size, and number of hash functions. Its ability to provide stable and accurate estimates, even with minimal computational resources, makes it the preferable choice for large-scale and resource-constrained environments.

- **Scalability:** G-LESE scales efficiently with the number of sets, overlap levels, and number of hash functions, ensuring reliable performance across diverse datasets without the need for extensive computational resources.
- **Resource Efficiency:** In scenarios with limited computational power or low hash-functions counts, G-LESE demonstrates superior accuracy and stability compared to JISE, which struggles under similar conditions.
- **Robustness:** G-LESE's performance remains unaffected by overlap and set size, providing a robust solution that adapts to varying dataset conditions without significant performance degradation.
- **Consistency:** While JISE shows improvement only at higher amount of hash functions, G-LESE consistently outperforms JISE at all hash-functions levels, overlap rates, and set sizes, making it the algorithm of choice for most applications.

In conclusion, while JISE performs well in controlled environments with a high number of hash functions, its sensitivity to overlap and set size constrains its scalability. In contrast, G-LESE provides a more balanced, efficient, and reliable all-round solution.

5.6 Heuristic Insight: JISE

Despite the lower performance of JISE in comparisson with G-LESE , we noticed experimentaly a specific behavior of the JISE algorithm. Let:

$$\Delta = |A - B|,$$

where A is JISE's estimate with a low number of hash functions and B is JISE's estimate with a higher number of hash functions or the real cardinality of the union of all sets. Our observation is related to the overlap and to the number of hash functions that have been used, as these insights reveal critical performance variations of the JISE algorithm.

Before proceeding to the results that led us to consider the JISE approach as a potential heuristic methodology, we first introduce and visualize the concept of global overlap among multiple sets. To illustrate this, we present an example involving three sets with overlap values of 0.2, 0.5, and 0.8.

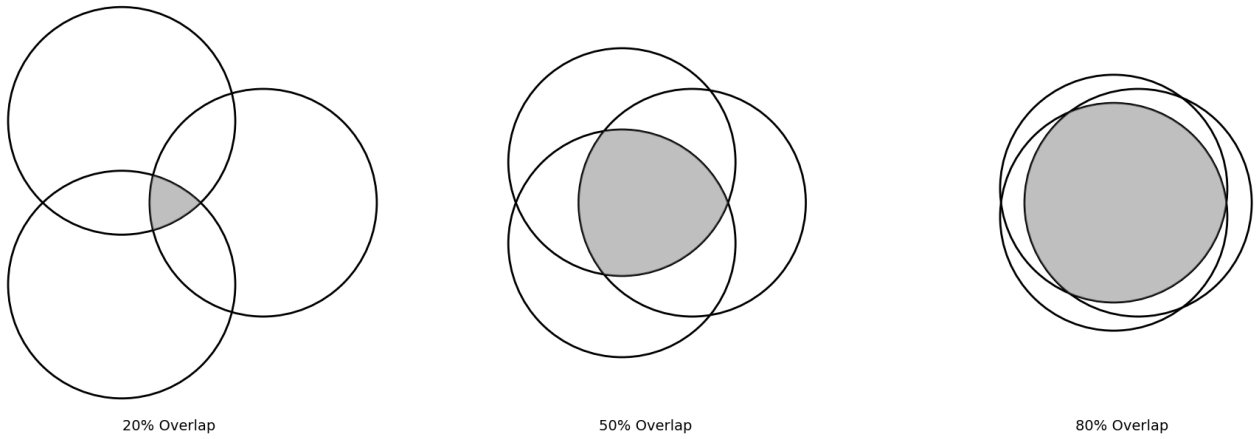


Figure 5.11: Overlap of 0.2, 0.5, and 0.8 for three sets.

Next, to provide further insight into how global overlap manifests visually, we extend the example to six and 10 sets.

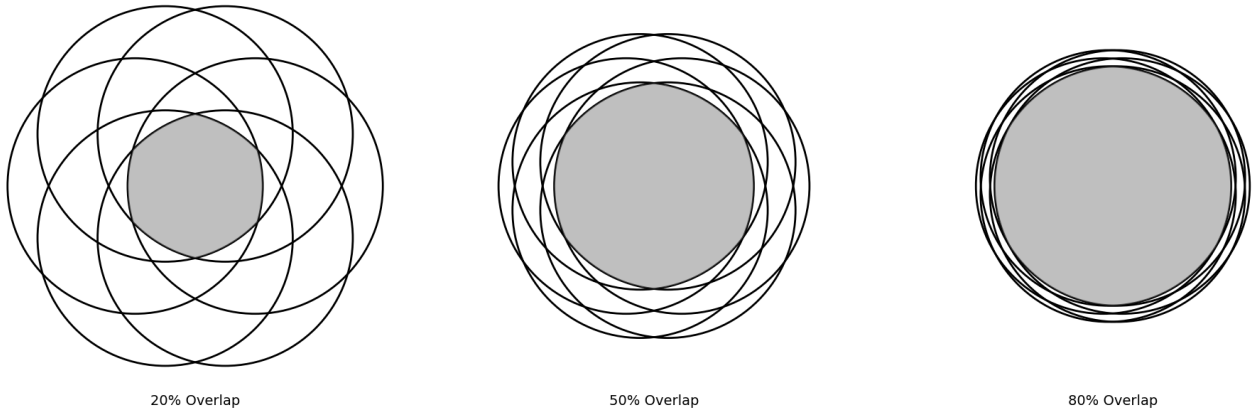


Figure 5.12: Overlap of 0.2, 0.5, and 0.8 for 6 sets.

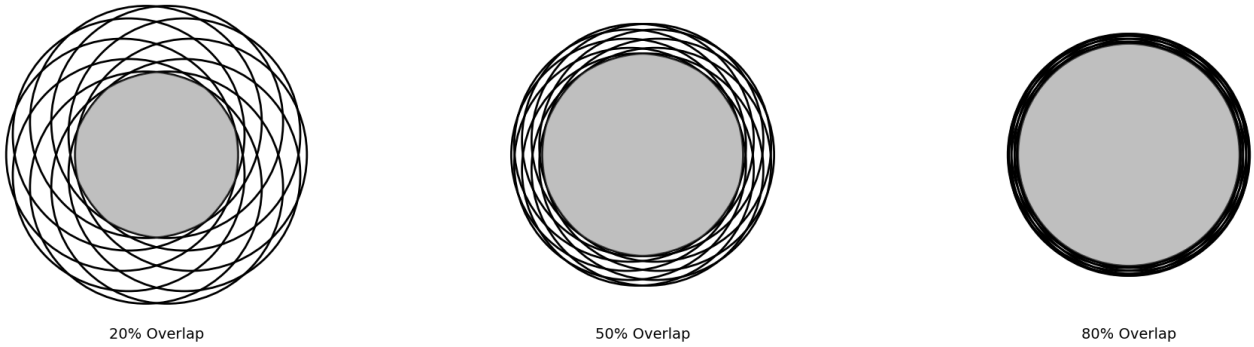


Figure 5.13: Overlap of 0.2, 0.5, and 0.8 for 10 sets.

We now present an example showing the average error of a single experiment, repeated with three random seeds, due to the probabilistic nature of selecting random hash functions, which may lead to minor anomalies. Furthermore, we demonstrate the interplay of the JISE algorithm with respect to different global overlap values.

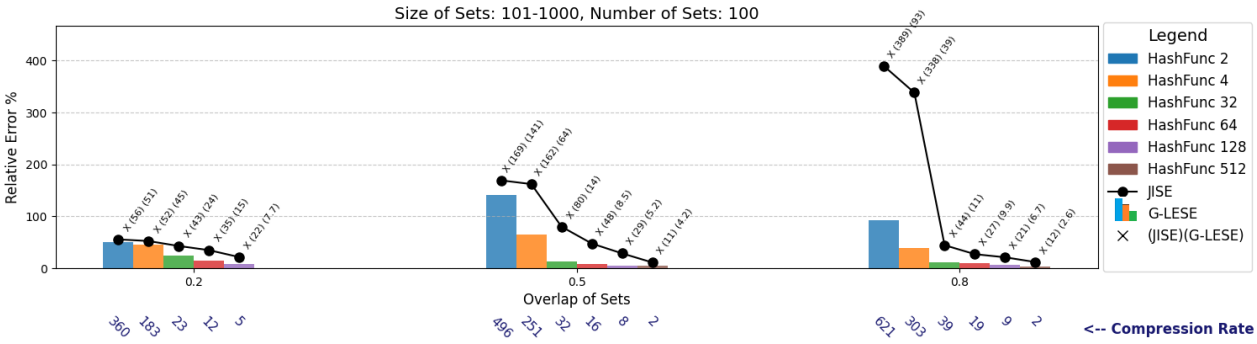


Figure 5.14: Heuristic Insight: Overlap 0.2 , 0.5 , 0.8. - average of 3 different seeds

At low to medium number of hash functions, Δ grows significally when overlap is getting bigger and bigger.,For low-overlap datasets, Δ remains small as JISE manage to "loose" the few /low similarities , and the estimation apporaches the real caridnality. Thus, Δ serves as a practical measure of overlap:

$$\Delta \propto \text{Global Overlap.}$$

where

$$\Delta = |\text{JISE(low number of hashFunc)} - \text{JISE(high number of hashFunc)}|$$

Larger Δ suggests higher overlap, while smaller Δ indicates minimal redundancy, leveraging the error differences seen in the graphs. We continue now , presenting and emphasizing the impact of overlap on the JISE algorithm by analyzing it under different set's sizes and number of sets. First we will continue the plots with avg of the 3 seeds for all cases, and then with a single seed running at once all the experiments, introducing randomness on selecting correct hash functions.

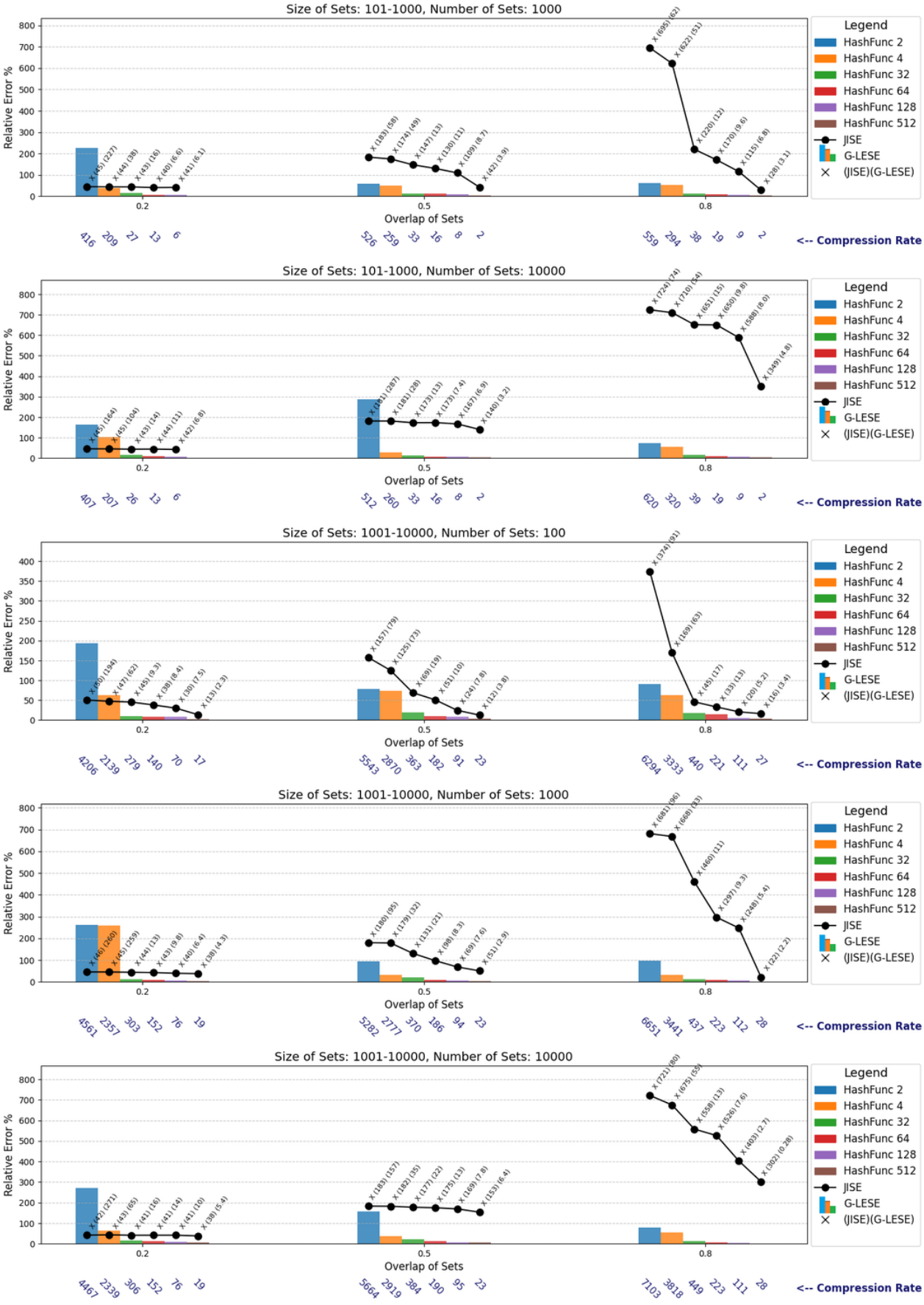


Figure 5.15: Heuristic Insight: for Overlap 0.2, 0.5 and 0.8. - average of 3 random different seeds

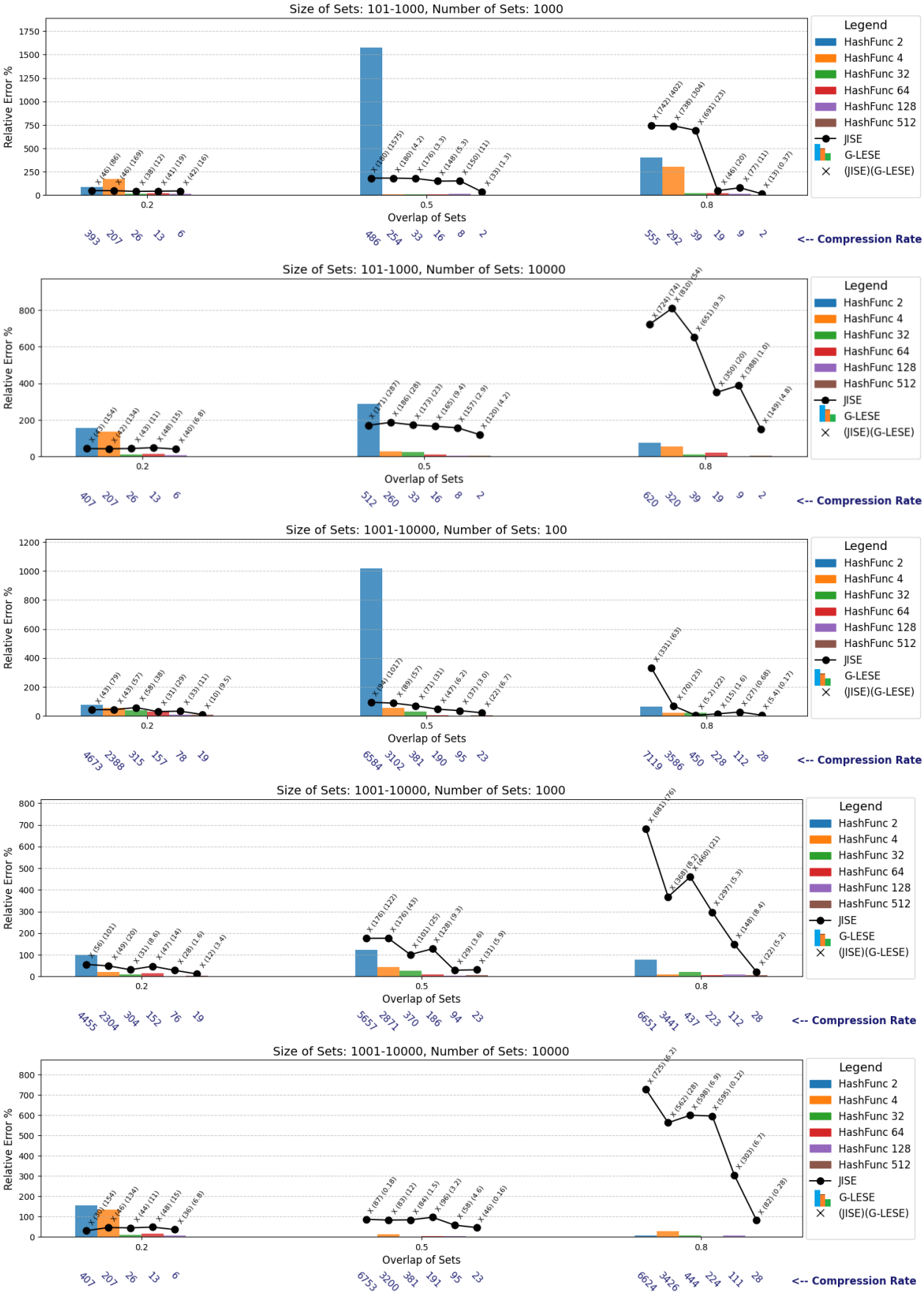


Figure 5.16: Heuristic Insight: for Overlap 0.2, 0.5 and 0.8. - 1 random seed

5.7 Transmitting Approaches

Counter Cardinality Minhash (CCM) and Full Stack Cardinality Minhash (FCM) are two complementary techniques for transmitting sensor data and estimating cardinalities. Below, we visualize how data is transmitted in both techniques, as well as how HoMoEdges are identified. CCM includes a counter for direct object counts, whereas FCM relies on cardinality estimation based on minhash signatures alone.

Leader Node (L_j)	**Data Set** (O_{e_i})
e_i	{MinHash Sig.$_{e_i}$}
⋮	⋮
e_n	{MinHash Sig.$_{e_n}$}

Table 5.4: Data Transmission in FCM

Leader Node (L_j)	**Data Set** (O_{E_i})
e_i	{$Counter_{e_i}$, MinHash Sig.$_{e_i}$}
⋮	⋮
e_n	{$Counter_{e_j}$, MinHash Sig.$_{e_n}$}

Table 5.5: Data Transmission in CCM

We delineate the relationship between the data transmission techniques—Counter Cardinality MinHash (CCM) and Full Stack Cardinality MinHash (FCM)—and the algorithms employed to estimate set cardinalities and identify HoMoEdges and HoMoPaths. The following table encapsulates these connections.

Technique	**Estimate Sensor Cardinality**	**Estimate Union Cardinality**	**Homoedges**	**Homopaths**
CCM	Counter	G-LESE or JISE	Counter/G-LESE or Counter/JISE	Jaccard Index Similarity
FCM	A-LESE	G-LESE or JISE	A-LESE/G-LESE or A-LESE/JISE	Jaccard Index Similarity

Table 5.6: Algorithmic Approaches Aligned with CCM and FCM Techniques

Table 5.6 summarizes the data transmission techniques (CCM and FCM) and their corresponding algorithms for union cardinality estimation, sensor cardinality, and detecting HoMoEdges and HoMoPaths. CCM uses counters with G-LESE or JISE to provide exact atomic sizes and estimated global sizes, limiting estimation uncertainty to the global scale. In contrast, FCM relies entirely on estimation methods for both atomic and global sizes, using MinHash signatures to estimate cardinalities and identify HoMoEdges and HoMoPaths, without transmitting explicit counters.

Example CCM vs FCM

Consider a scenario where the actual distinct count of objects is 10,000. With an estimation error of ±10%, the estimated cardinality could either be 11,000 or 9,000. A threshold of 10% is used to determine the hotness of a sensor. If the ratio between the local cardinality and the global cardinality exceeds this threshold, the sensor is classified as "hot."

Real Cardinality	**Counter**	**G-LESE or JISE**	**Ratio (%)**	**HoMo Edge?**
10,000	1,000	11,000	9.09	Yes- Correct
10,000	1,000	9,000	11.11	No- Wrong

Table 5.7:
HoMoEdge Estimation in CCM

Real Cardinality	**A-LESE**	**G-LESE or JISE**	**Ratio (%)**	**HoMo Edge?**
10,000	1,100	11,000	10	Yes- Correct
10,000	900	9,000	10	Yes- Correct

Table 5.8:
Homoedge Estimation in FCM

Consider the same scenario, but A-LESE overestimates at 1,100 while G-LESE underestimates at 9,000. In this case , we will successfully detect the HoMoEdge , which is well above the threshold, but this misaligned will lead to false positives HoMoEdges and to an overhead in the network communication costs. Similary, if A-LESE underestimates at 900 while G-LESE overestimates at 11,000, then we incorrectly exclude a HoMoEdge, leading to false negatives, resulting in information loss for identifying motion paths.

Real Cardinality	**A-LESE**	**G-LESE**	**Ratio (%)**	**HoMoEdge?**
10,000	1,100	9,000	12.22	Yes- Correct - Increases False Positive
10,000	900	11,000	8.18	No- Wrong - Increases False Negative

Table 5.9:
Example of FCM Failure Due to Misaligned Estimation Errors

5.7.1 Hot Motion Edge Detection via CCM

In this work, we employ the Cardinality Counting Method (CCM) to identify Hot Motion Edges (HoMoEdges) in a transportation network. The primary objective of this approach is to evaluate the relative importance of each edge in terms of the traffic it supports. Specifically, we ensure that the numerator in our formula 4.3 represents the true traffic volume, as reported by sensors. This local traffic count accurately reflects the real conditions on each edge.

The denominator, on the other hand, is estimated as the global cardinality of traffic across the entire network. This global estimate provides a holistic view of the overall traffic volume. By comparing the local count to the global estimate, we identify edges that meet or exceed a specified threshold, classifying them as HoMoEdges. This enables efficient detection of traffic hotspots in complex networks.

Below, we present the HoMoEdge Detection algorithm using the CCM approach:

Algorithm 5 HoMoEdge Detection (HOMOED - CCM)

Require: $Edges$: List of all edges $e_1, e_2, ..., e_n$, $Threshold$: Minimum support threshold
Ensure: $HoMoEdges$: List of detected hot motion edges

$HoMoEdges \leftarrow \emptyset$
for all $e \in Edges$ **do**
    $GlobalEstimate \leftarrow$ G-LESE($Edges$)
    $LocalCount \leftarrow$ GetLocalCount(e)
    if $\frac{LocalCount}{GlobalEstimate} \geq Threshold$ **then**
        $HoMoEdges \leftarrow HoMoEdges \cup \{e\}$
    end if
end forreturn $HoMoEdges$

5.8 Unified Algorithm for Hot Motion Edge and Path Detection (HoMoPaD)

Building upon the individual methodologies of HoMoEdge Detection and HoMoPath Detection, we propose a unified HoMoPaD algorithm to streamline the detection process for both edges and paths, referred to as the Unified HoMoEdge and HoMoPath Detection algorithm, considering edges as path with length 1.

This integrated approach leverages the strengths of both methods to deliver a more holistic understanding of motion patterns within a network. It operates in two key stages:

1. **Hot Motion Edge Detection:** The algorithm begins by identifying HoMoEdges using the CCM approach, ensuring that edges with significant traffic, relative to global estimates, are detected.

2. **Hot Motion Path Detection:** After detecting HoMoEdges, the algorithm extends its focus to identify HoMoPaths. These paths represent sequences of edges

forming contiguous routes with notable motion patterns. MinHashing and Jaccard similarity metrics are employed to evaluate the significance of each path.

The unified algorithm is presented as follows:

Algorithm 6 Unified HoMoEdge and HoMoPath Detection (U-HoMoPaD)

Require: $Edges$: List of all edges $e_1, e_2, ..., e_n$, $Threshold_E$: Minimum support threshold for edges, $Threshold_P$: Minimum Jaccard similarity threshold for paths, $n = 2$ (minimum path length)

Ensure: $HoMoEdges$: List of hot motion edges, $HoMoPaths$: Dictionary of hot motion paths

$HoMoEdges \leftarrow \emptyset$
$HoMoPaths \leftarrow \emptyset$

Step 1: Detect Hot Motion Edges
for all $e \in Edges$ **do**
    $GlobalEstimate \leftarrow$ G-LESE($Edges$)
    $LocalCount \leftarrow$ GetLocalCount(e)
    if $\frac{LocalCount}{GlobalEstimate} \geq Threshold_E$ **then**
        $HoMoEdges \leftarrow HoMoEdges \cup \{e\}$
    end if
end for

Step 2: Detect Hot Motion Paths
$Length \leftarrow n$
while `Matrix Contains Non-Zero Values` **do**
    $HotPaths \leftarrow$ Initialize_Empty_List()
    for all `Path` p `on Horizontal Row (e.g.,` $p = \{e_1, e_2, ..., e_{n-1}\}$`)` **do**
        for all `Edge` q `on Vertical Column` **do**
            if `p[len(p)] is Connected to` q **then**
                $Signatures_p \leftarrow$ MinHash(p)
                $Signatures_q \leftarrow$ MinHash(q)
                $JaccardIndex \leftarrow$ Calculate_Jaccard_Index($Signatures_p, Signatures_q$)
                if $JaccardIndex \geq Threshold_P$ **then**
                    $HotPaths \leftarrow$ Add($p \cup q$, MinHash($Signatures_p \cap Signatures_q$))
                end if
            end if
        end for
    end for
    $HoMoPaths[Length] \leftarrow HotPaths$
    Remove_Subsets($HoMoPaths[Length]$)
    Update_Matrix($HoMoPaths[Length]$, BooleanMatrix)
    $Length \leftarrow Length + 1$
end whilereturn $HoMoEdges, HoMoPaths$

Description of the Unified HoMoEdge and HoMoPath Detection Algorithm

Lines 1-2: Algorithm Requirements and Outputs.
The algorithm, referred to as *Unified HoMoEdge and HoMoPath Detection (U-HoMoPaD)*, takes as input a Dictionary of the road connections, a minimum support threshold, and the transmitting data of each sensor - *Extra Counter* + *MinHashSignature* - as described in Section 5.4.1. The outputs are (i) *HoMoEdges*, and (ii) *HoMoPaths*.

Lines 3-4: Initialization of Data Structures.
HoMoEdges will be populated in **Step 1** to store edges that exceed the minimum support threshold, while *HoMoPaths* will group extended paths identified in **Step 2**, following the same threshold-based bridging principle. This bifurcated structure directly integrates *HoMoEdge Detection* and *HoMoPath Detection* into a single pipeline.

Lines 6-12: Hot Motion Edge Detection (Step 1).
For every edge e in the global list of edges, the algorithm first invokes the G-LESE function (Line 7). This call estimates the global traffic volume as described in Section 5.4.3. Next, the local count (i.e., the observed unique object count) for each edge is obtained with GetLocalCount(e), leveraging the CCM approach with each sensor sending the *Extra Counter*. If the ratio of the local count to the global estimate meets or exceeds $Threshold_E$, the edge e is appended to the *HoMoEdges* list (Line 11).

Lines 14-15: Transition to Hot Motion Path Detection.
Once potential *HoMoEdges* have been identified, the algorithm sets $Length \leftarrow n$ (Line 15). By default, $n = 2$, so the next phase begins by considering paths formed by combining single edges (length-1) with an additional edge.

Lines 16-29: Hot Motion Path Detection (Step 2).
To detect *HoMoPaths*, the algorithm employs a *BooleanMatrix* to iterate through possible path extensions, as detailed in Section 5.3 and illustrated in Algorithm 1 with an extended example. Here, each row corresponds to an existing HoMoPath, while each column represents a HoMoEdge that can extend at least one of the remaining HoMoPaths, based on our pruning logic, as described in Example 5.3.2. Whenever the last edge of a HoMoPath is connected to a HoMoEdge (see *Step 2* details), the algorithm computes their respective MinHash signatures and evaluates the resulting Jaccard similarity. Paths whose similarity meets or exceeds *Threshold* are extended and added to the *HotPaths* list. Subsequently, Remove_Subsets prunes redundant paths entirely contained within larger paths, and Update_Matrix refreshes the *BooleanMatrix* to reflect newly created paths. The path length *Length* is then incremented to explore longer sequences. The iteration terminates when no further extensions are possible, yielding a comprehensive set of hot motion edges (*HoMoEdges*) and extended paths (*HoMoPaths*) for a unified network view.

Chapter 6

Experimental Results

This chapter evaluates our Hot Motion Paths Detection (*HoMoPaD*) methodology in large-scale road networks, quantifying the accuracy and cost-efficiency of MinHash signatures for compressed data transmission. We analyze trade-offs between compression, accuracy, and first-level communication costs between sensors and Regional Leaders, building on Chapters 3 and 4.

Experiments were conducted on California, Munich, and Chania road networks, each spanning ∼200 edges and divided into four regions under a decentralized hierarchy. Performance is measured via Jaccard similarity, comparing *HoMoPaths* from raw tagIDs and MinHash signatures. This chapter also explores the challenges of data compression in sensor networks with non-uniform traffic and presents the scientific rationale behind our approach. The corresponding code is available at the following link: `https://github.com/christodouloskampanis/HoMoPaD`.

6.1 Experimental Setup

6.1.1 Data Collection and Network Preparation

Real-world road network data was extracted using the `osmnx` library, which provides detailed representations of road segments and intersections. The three networks analyzed in this study are:

1. **Munich, Bavaria, Germany:** Located in the heart of Munich, this region represents a typical European road network layout.

2. **Outer Sunset, San Francisco, CA, USA:** Part of the San Francisco Network, this area features a densely connected road structure with a significant number of intersections.

3. **Chania, Crete, Greece:** A popular tourist destination in Greece, characterized by a dense road network and high traffic volumes, especially during the summer months.

6.1.2 Regional Partitioning

The network is recursively partitioned into n non-overlapping regions, $R_{g1}, R_{g2}, \ldots, R_{g_n}$, forming a decentralized structure where each operates under a Regional Leader ($RegL$). This hierarchy optimizes data aggregation and scalability, while enhancing adaptability and efficient load distribution, making it ideal for real-world applications.

Example of Partitioning Munich Network :

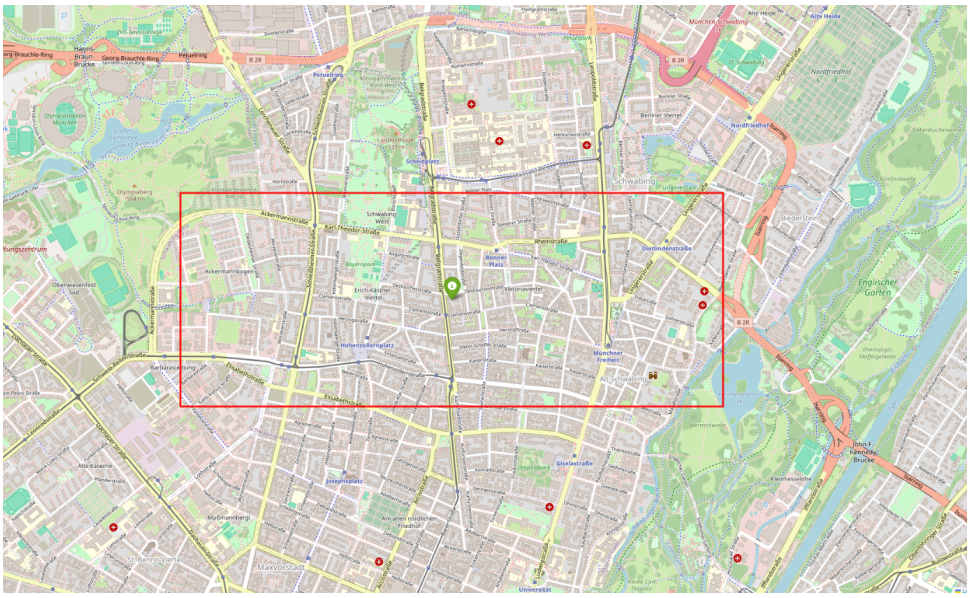


Figure 6.1: Area in Munich with 180 Edges

The network is modeled as a graph. Due to unknown or outdated(due to its dynamic nature) edge direction, all edges are assumed bidirectional. The next Figures present the partitioning process, outlined in Chapter 4.

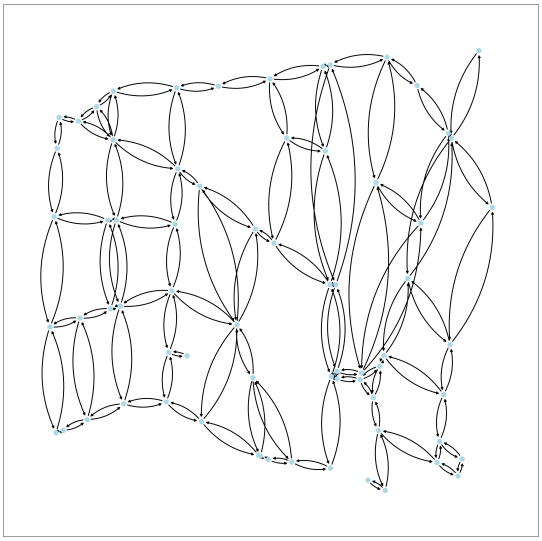
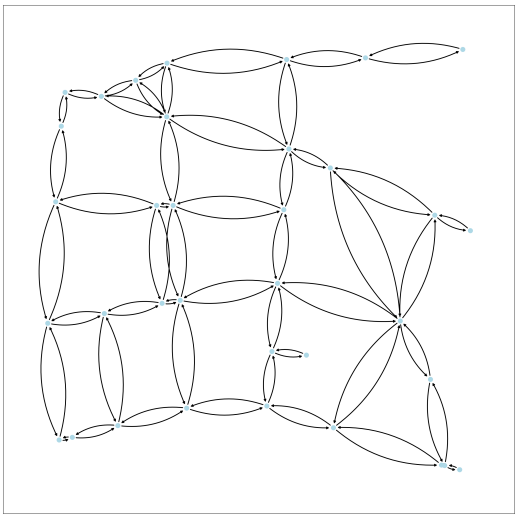
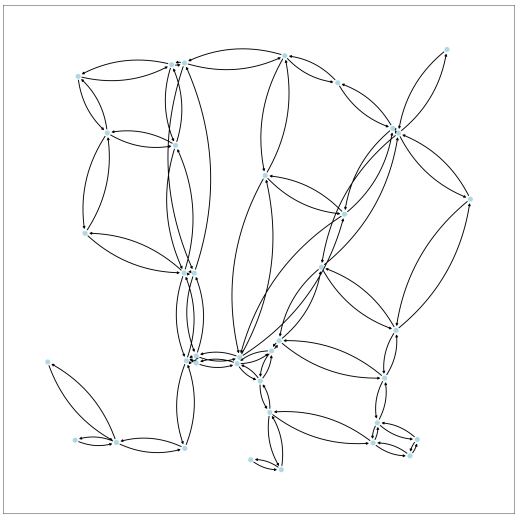


Figure 6.2: Area in Munich with 180 Edges

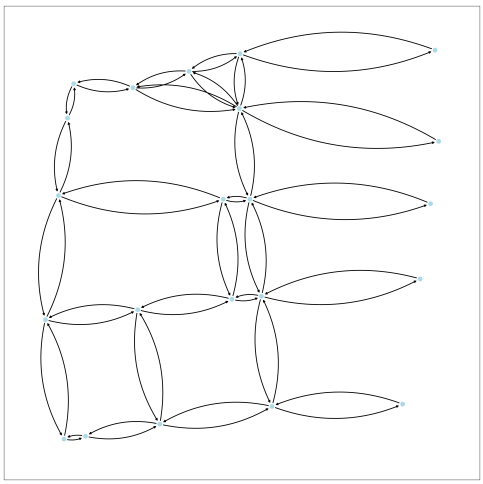


(a) Region 1.

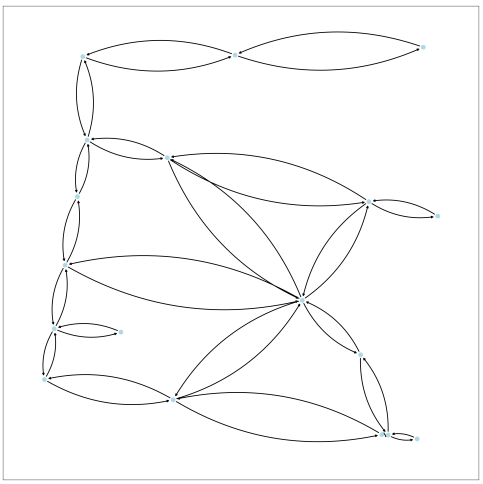


(b) Region 2.

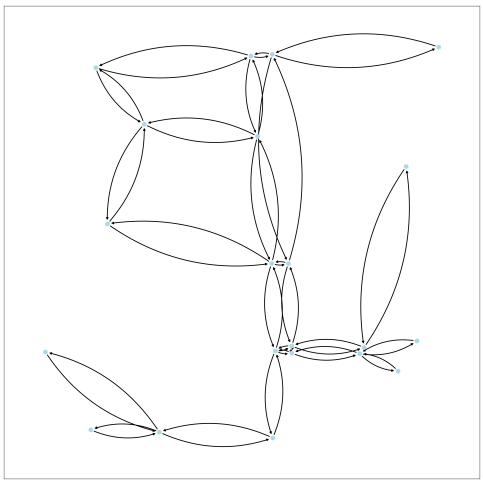
Figure 6.3: 2-Partitioning : 2 Regions (R_{g1}, R_{g2}).



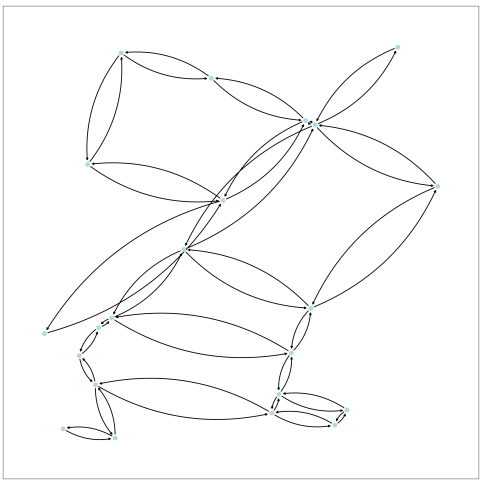
(a) Region 1



(b) Region 2



(c) Region 3



(d) Region 4

Figure 6.4: 4-Partitioning : 4 Regions ($R_{g1}, R_{g2}, R_{g3}, R_{g4}$).

6.1.3 Simulating Traffic:

To approximate real-world traffic behavior, we generated moving objects (tagIDs) traversing the network edges. These movements were simulated as random walks with varying path lengths to ensure heterogeneous traffic flow. The total number of tagIDs was varied from 1000 to 25, 000, enabling us to examine the impact of traffic volume on detection accuracy.

6.1.4 Accuracy Metric: Jaccard Similarity

To compare the accuracy of *HoMoPaD* detection between raw and compressed datasets, we use the Jaccard similarity metric, defined as:

$$J_{\text{similarity}} = \frac{|P_{\text{raw}} \cap P_{\text{minhash}}|}{|P_{\text{raw}} \cup P_{\text{minhash}}|},$$

where:

- P_{raw}: The set of *HoMoPaths* detected using raw tagIDs.
- P_{minhash}: The set of *HoMoPaths* detected using compressed MinHash signatures.

The Jaccard similarity measures the overlap between these two sets, with values closer to 1 indicating higher accuracy. This metric is especially suited for our study, as it incorporates both precision and recall in the detection process. Our approach involves comparing P_{raw} and P_{minhash} as if we simply color the graphs to visualize the *HoMoEdges* and show this to the user as a graphical depiction.

6.1.5 Communication Costs 1st level- TOSSIM

To evaluate the unified algorithm, we utilized **TOSSIM** [27], a TinyOS-based simulator for wireless sensor networks. This allowed us to simulate communication costs associated with executing the algorithm in a networked environment. Specifically, we measured the total number of bytes transmitted, emphasizing also on the NACK bytes due to collisions. The experiments were conducted under Best, Worst and Average Case for each experiment , under numerous repetitions. This analysis provided valuable insights into the algorithm's efficiency and practicality for real-world applications.

6.2 Munich Network - HoMoED (CCM)

6.2.1 Accuracy

For a total traffic volume of 1000 tagIDs, Figure 6.5 illustrates the precision of the Hot Motion Edge Detection phase. The horizontal axis represents the threshold, while the vertical axis shows the accuracy. The legend provides details on compression levels and the corresponding number of hash functions. For instance, the blue bars indicate compression level 4, achieved using 10 hash functions.

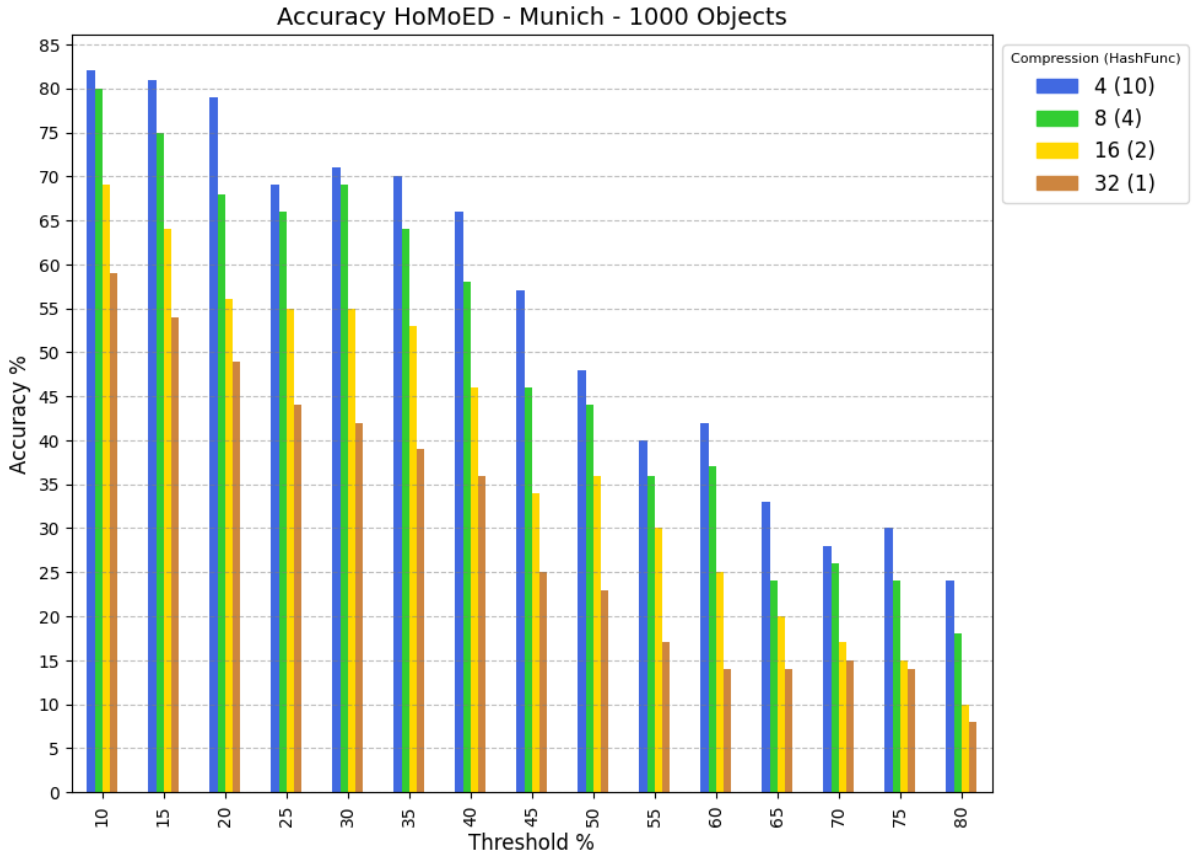


Figure 6.5: Accuracy - Munich network, 180 sensors, 1000 tagIDs.

6.2.2 Communication Costs 1st Level - Tossim

Along with accuracy, it is equally important to evaluate the associated communication costs at varying compression levels. Reduced compression improves accuracy but increases communication overhead. Thus, we face a trade-off between accuracy and communication costs. We start by analyzing the initial communication level, which occurs between the sensors and the Regional Leaders (RL_i).

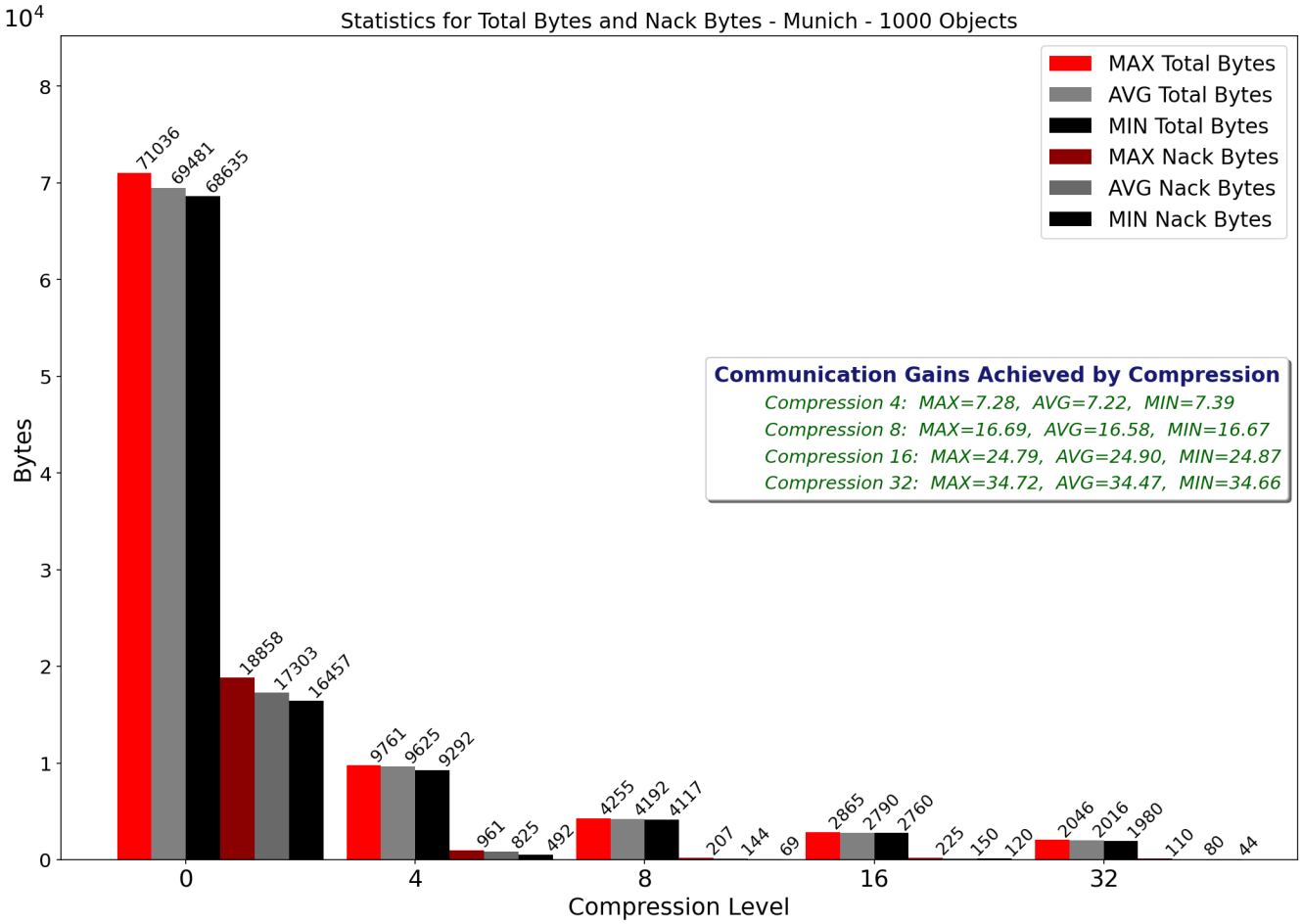


Figure 6.6: Communication Costs 1st Level - Munich network, 180 sensors, 1000 tagIDs.

In the initial experiments described in 6.1.5, we measured the maximum, average, and worst-case scenarios to assess how varying compression levels influence communication costs. To these measurements, the plots labeled "Communication Gains Achieved by Compression" visualize how varying compression rates translate into network-wide cost reductions. Notably, at a compression level of 4, the data is reduced by a factor of 4, yet the communication costs drop by a factor of 7.22. This result effectively doubles the expected savings and underscores the potential for substantial gains when compressing sensor data. Similarly, at a compression level of 8, the communication costs are reduced by a factor of 16.58, illustrating that higher compression ratios can yield even more pronounced improvements. These gains highlight the non-linear relationship between the theoretical compression factor and the actual reductions in communication overhead. This difference stems from the reduction in NACKs, resulting from fewer collisions and, consequently, fewer message retransmissions. In the following plots, we illustrate the non-linear behavior and note the limited gains under very high traffic volumes. Additionally, the overall data volume depends on the chosen time-window size and the fine-tuned spatial distribution of the road network.

6.2.3 Accuracy and Communication Costs

Since our methodology relies on compression, different traffic volumes combined with the same compression level produce MinHash signatures of varying length. Larger traffic volumes yield more data for compression, which can affect signature size. Consequently, communication costs may vary even under identical compression settings. To test scalability, we repeated experiments with traffic volumes of 2,500 , 5,000 , 10,000 , and 25,000 tagIDs. By doing this, we assessed how the algorithm behaves under various traffic loads, acknowledging that even within the same network, traffic volumes may differ significantly at different times of the day.

Number of Objects = 2,500

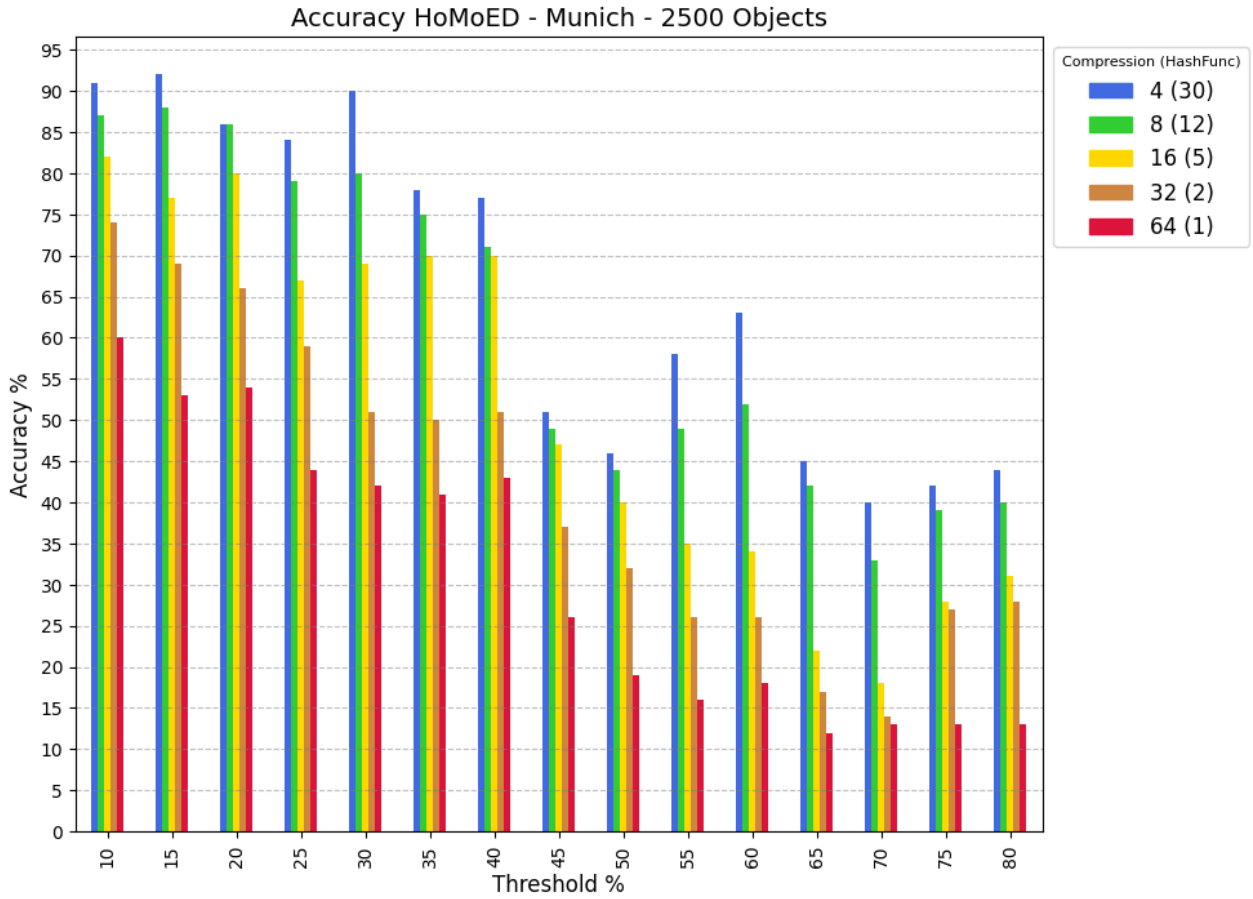


Figure 6.7: Accuracy - Munich network, 180 sensors, 2.5k tagIDs.

For a compression ratio of 4:1, as observed in Figures 6.7 and 6.8, accuracy remains well-preserved up to a threshold of approximately 40%, maintaining a minimum accuracy of 80%. This level of compression leads to significant communication gains, with a gain factor of 9.2 (see Table *"Communication Gains Achieved by Compression"* in Figure 6.8). Notably, even at this relatively low compression ratio, the transmission gain exceeds and

in some cases doubles the data compression rate, demonstrating the effectiveness of our approach in reducing communication overhead beyond mere data size reduction.

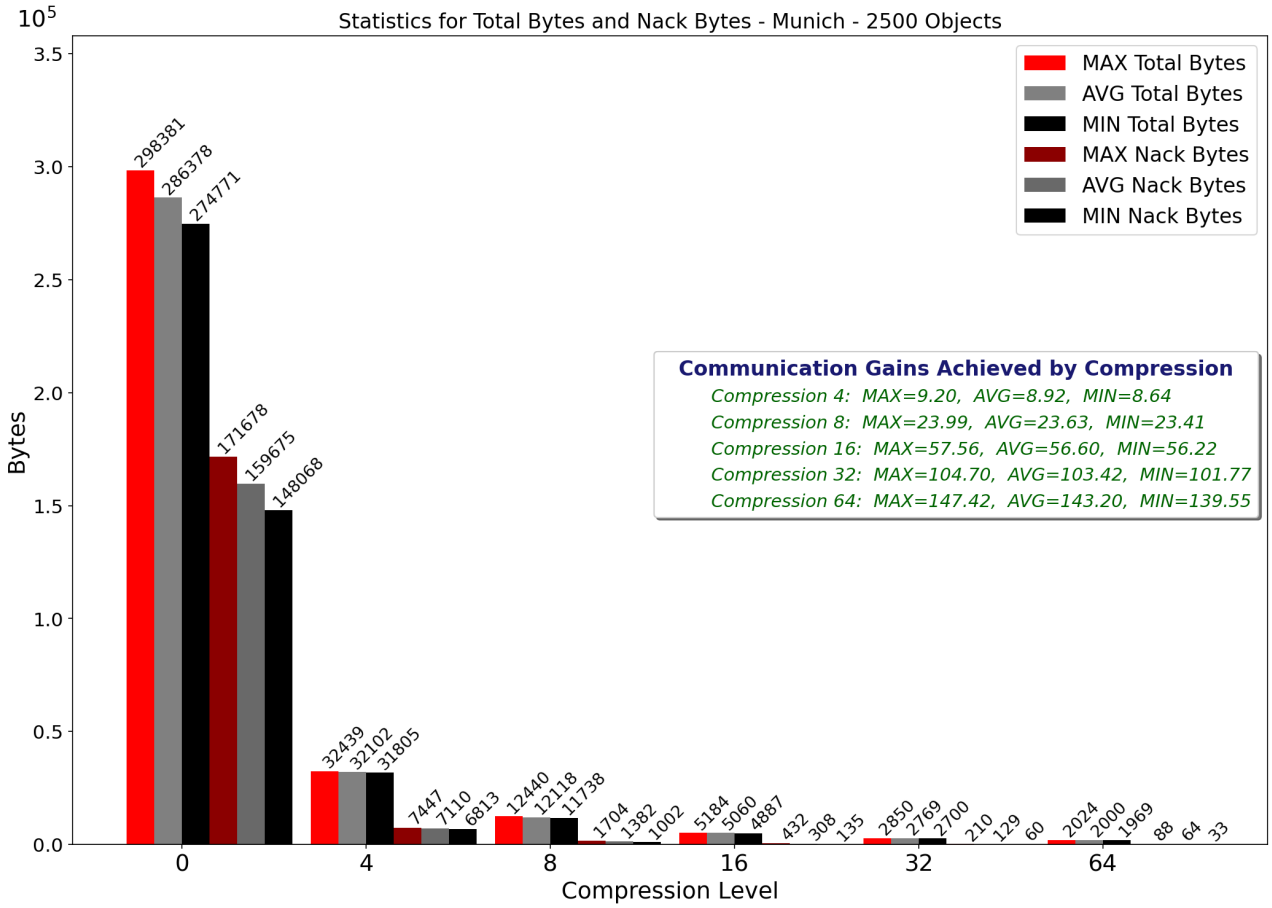


Figure 6.8: Communication costs 1st Level - Munich network, 180 sensors, 2.5k tagIDs.

A similar pattern emerges for a data compression ratio of 8:1, where accuracy remains at 70% for thresholds up to approximately 40%. In this case, the additional communication gain becomes even more pronounced, reaching a gain factor of 24, which is three times greater than the data compression ratio itself. These findings further highlight that data reduction not only minimizes storage and transmission costs but also significantly enhances communication efficiency by reducing network congestion and improving overall resource utilization.

However, for data compression ratios exceeding 8:1, accuracy drops significantly, making further compression less beneficial. This suggests that while higher compression ratios may lead to reduced data storage and transmission costs, they come at the expense of accuracy, indicating an optimal trade-off between compression efficiency and data fidelity.

Number of Objects = 5,000

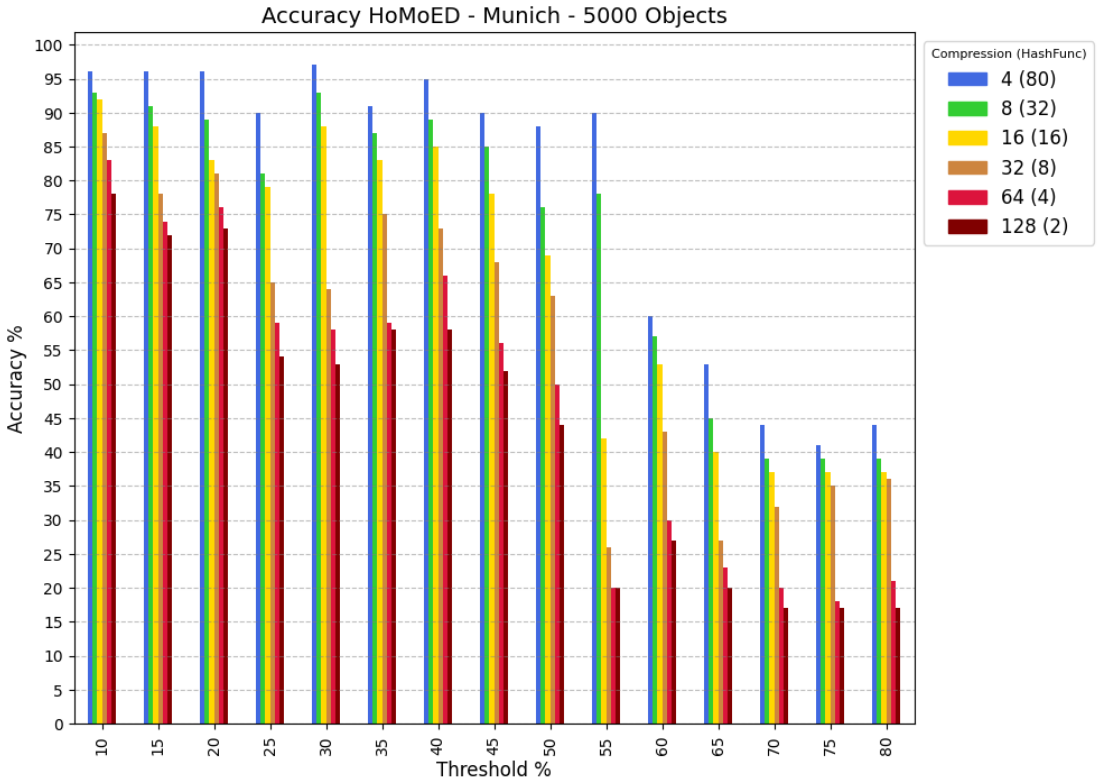


Figure 6.9: Accuracy - Munich network, 180 sensors, 5k tagIDs.

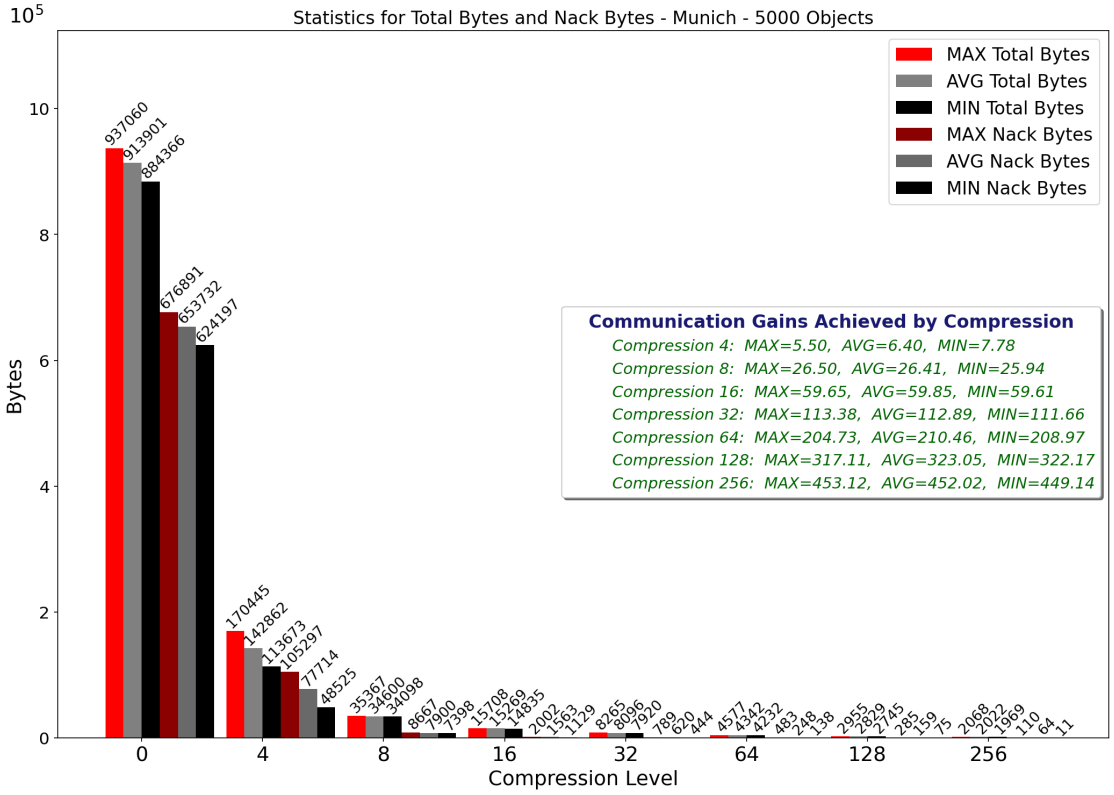


Figure 6.10: Communication costs 1st Level - Munich network, 180 sensors, 5k tagIDs.

As observed in Figures 6.9 and 6.10, the non-linear gains are evident. For a compression ratio of 4:1, we observe a remarkable accuracy preservation up to a threshold of approximately 50%, maintaining an accuracy of 90%. This level of compression results in significant communication gains, with a gain factor of 6.4 (see Table *"Communication Gains Achieved by Compression"* in Figure 6.10). Notably, even with such a relatively low compression ratio, the transmission gain surpasses the compression rate, demonstrating the effectiveness of our approach in reducing communication overhead beyond mere data size reduction.

A similar pattern emerges for a data compression ratio of 8:1, where accuracy remains at 80% for thresholds up to approximately 50%. However, in this case, we achieve a communication gain factor of 26:1, which is more than three times higher than the data compression ratio itself. For data compression ratios ranging between 16:1 and 32:1. and thresholds up to 45% , our method consistently maintains an accuracy of approximately 75%. Remarkably, these compression rates lead to substantial improvements in communication efficiency, with observed communication gain ratios of 60:1 and 112:1, respectively. This translates to an almost fourfold improvement in both cases compared to the data compression ratio alone, highlighting the significant impact of our methodology on data transmission efficiency. For higher compression ratios, the accuracy remains approximately 75% only for threshold values up to 20%.

Number of Objects = 10,000

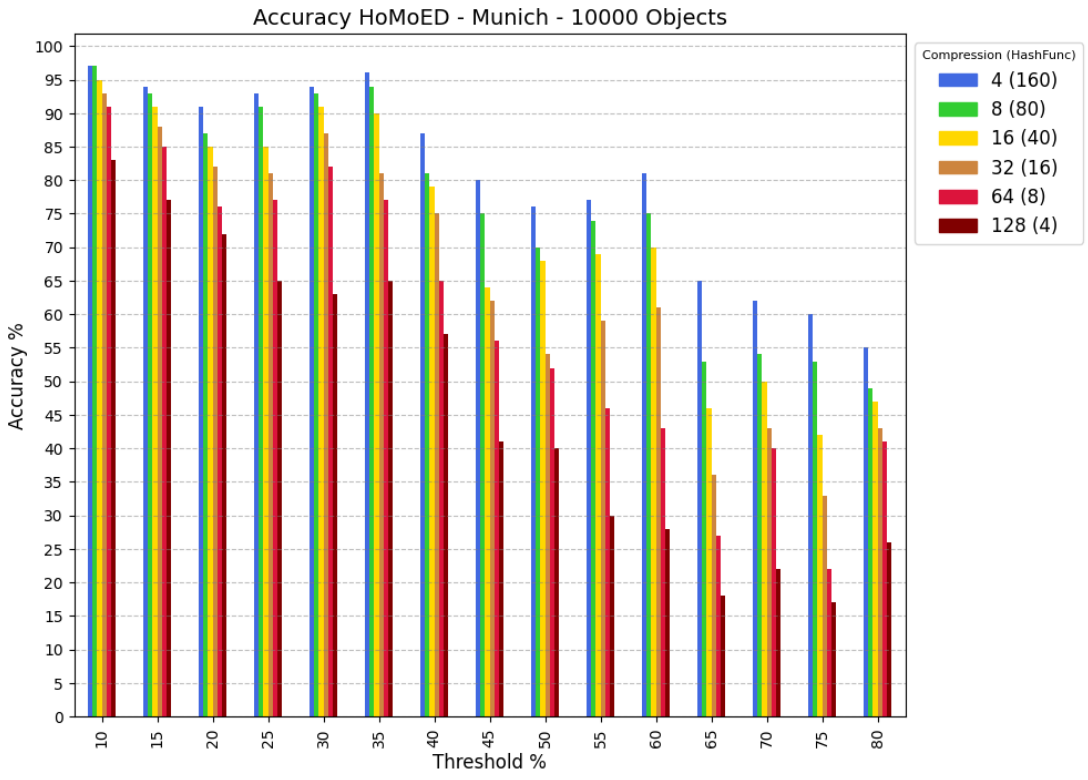


Figure 6.11: Accuracy - Munich network, 180 sensors, 10k tagIDs.

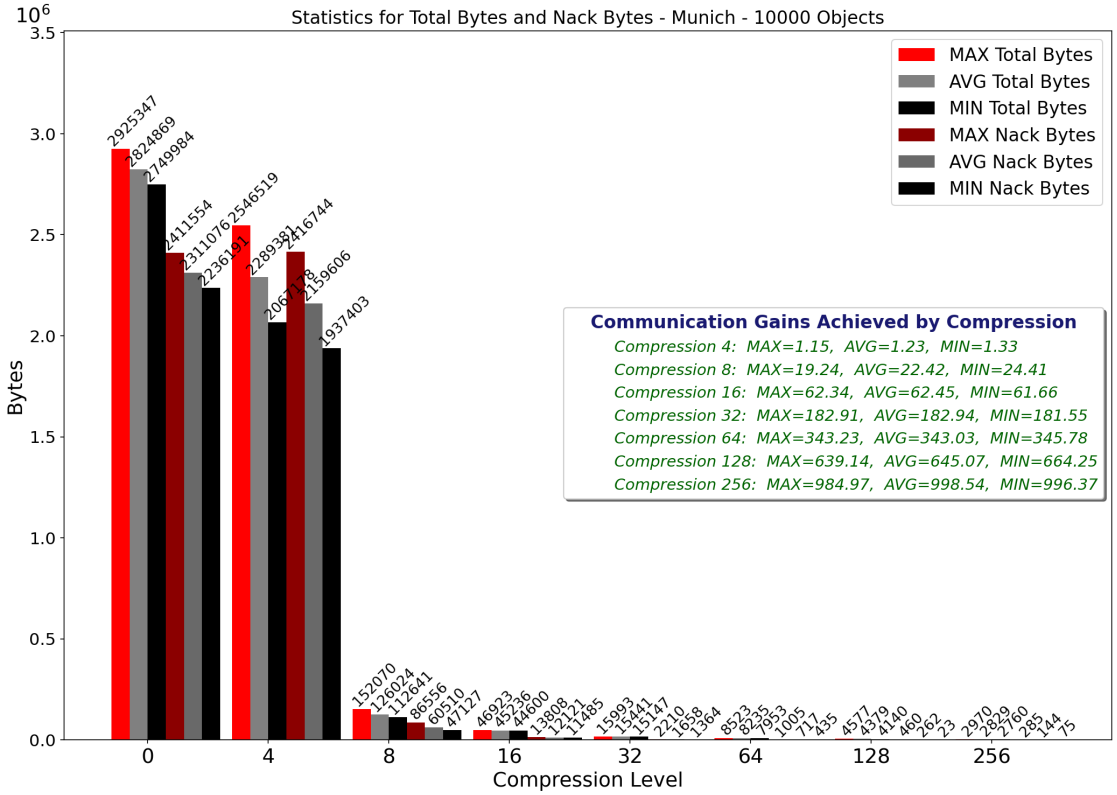


Figure 6.12: Communication costs 1st Level - Munich network, 180 sensors, 10k tagIDs.

From the table labeled "Communication Gain Achieved by Compression' in the Figure 6.12, one can observe that at a compression level of 4, the measured communication gain is only 1.23 for average and 1.15 and 1.33 for worse and best case.

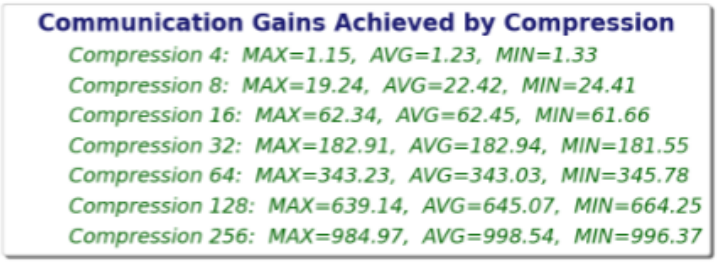


Figure 6.13: Communication Gain Achieved by Compression, 10K Objects

This value falls short of the theoretical factor of 4, indicating that reducing the data does not always translate into an equivalent reduction in communication overhead. However, starting from a compression level of 8, the observed communication gains begin to increase substantially, approaching or even surpassing their respective compression factors. This behavior highlights the fact that practical considerations, such as network overhead and data distribution, can lead to deviations from the idealized compression-versus-gain relationship.

With this exception in mind, the overall behavior remains consistent with the previously observed traffic volumes. For a data compression ratio of 8:1, accuracy remains close

to 75% for thresholds up to approximately 60%, achieving a communication gain factor of 22:1—almost three times higher than the data compression ratio itself. For compression ratios around 16:1, we observe a communication gain nearly four times greater than the compression rate, with accuracy maintained at approximately 70% for thresholds up to 60%.

At a compression ratio of 32:1 and thresholds up to 40%, our method consistently preserves an accuracy of approximately 75%, while simultaneously achieving a remarkable communication gain of 132:1, which is more than six times the data compression ratio.

Similarly, for higher compression ratios of 64:1 and 128:1, we observe communication gain factors of 343:1 and 645:1, respectively, which correspond to approximately 4–5 times the data compression ratio. Despite these high compression levels, accuracy remains at least 70% for thresholds up to nearly 40%. At compression ratio of 256:1, the accuracy remains acceptable only for limited thresholds up to 20%, indicating that beyond this point, the trade-off between compression and accuracy becomes increasingly unfavorable.

Number of Objects = 25,000

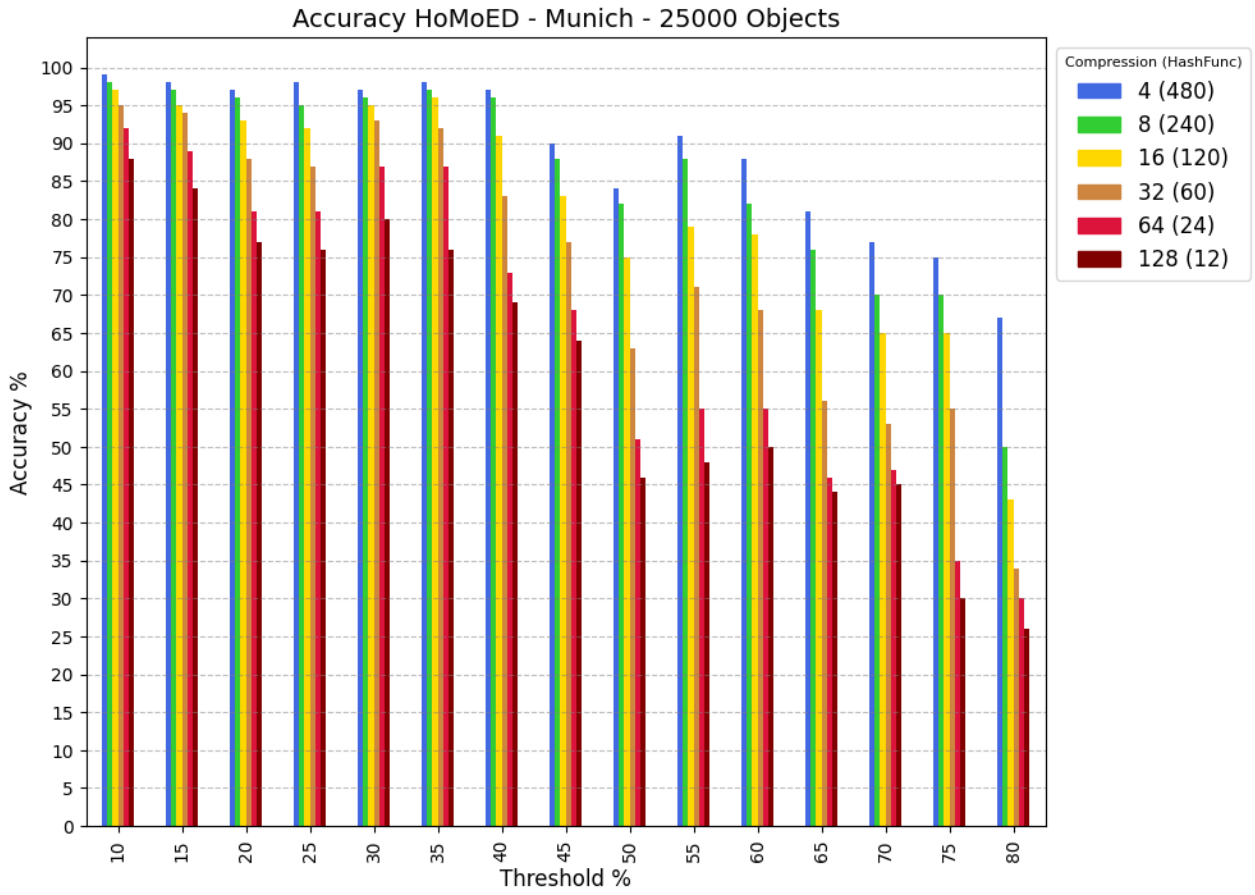


Figure 6.14: Accuracy - Munich network, 180 sensors, 25k tagIDs.

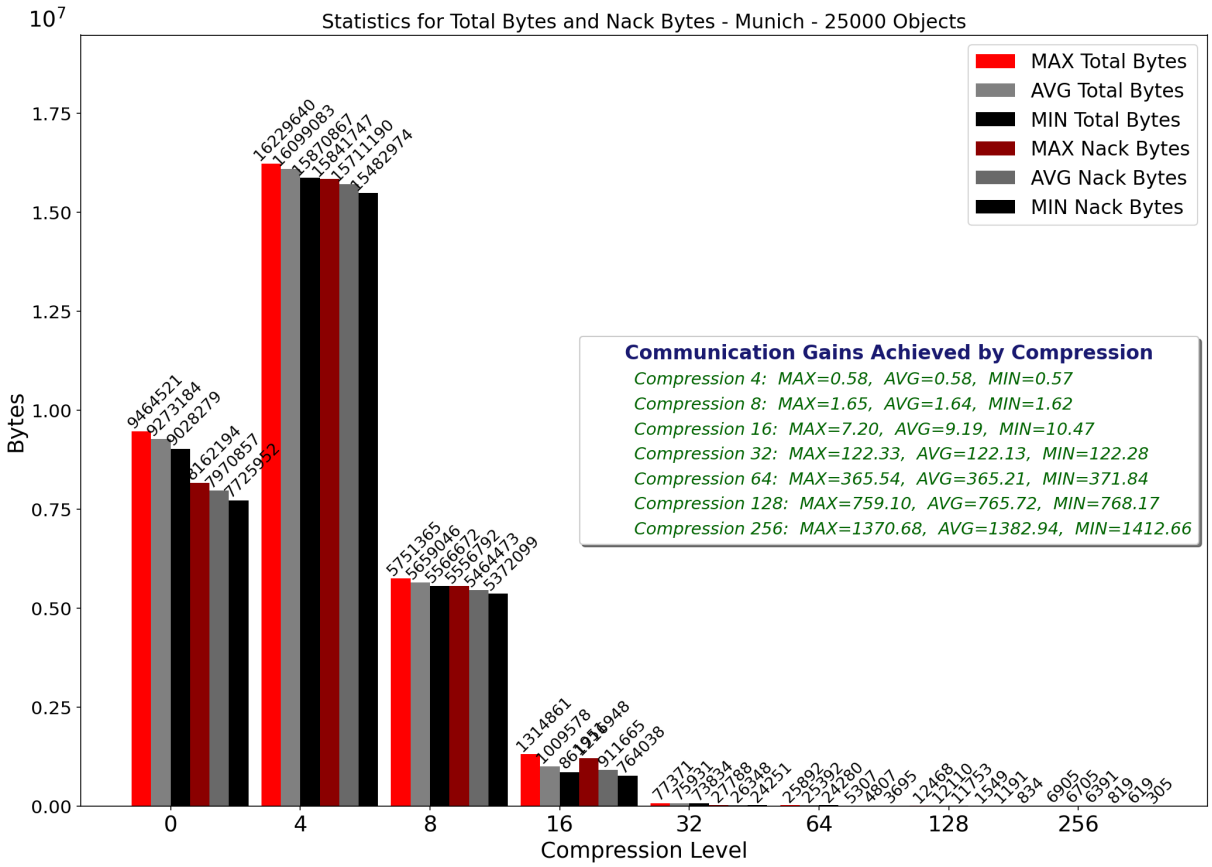


Figure 6.15: Communication costs 1st Level - Munich network, 180 sensors, 25k tagIDs).

An intriguing result emerges when scaling up to 25,000 objects: instead of achieving the anticipated reduction in communication costs, compression may actually increase overhead. As shown by the bars in Figure 6.15(and reflected in the table "Communication Gain Achieved by Compression"), the total cost at a compression factor of 4 exceeds that of the raw data. Specifically, the communication gain falls below 1 (0.58), indicating that the raw data requires approximately 42% fewer transmission resources compared to the supposedly compressed version. Analysis reveals that the primary factor driving this counterintuitive outcome is the significant increase in collision-related NACK bytes. Although compressing by a factor of 4 reduces the payload size, the elevated collision rate negates much of the benefit, leading to higher overall transmission costs.

Further insight into this phenomenon is provided by the uneven distribution of recorded objects among sensors. In scenarios without compression, sensors with fewer objects can transmit their data quickly, freeing network bandwidth for sensors with larger data volumes. However, once hashing techniques are applied, each sensor generates a uniform-length MinHash signature, regardless of how many objects it originally recorded. While this uniformity facilitates data comparison, it eliminates the natural advantage of having fewer objects to send, thereby increasing collisions and the associated overhead. This event underscores the nuanced trade-offs between compression ratios, collision dynamics, and real-world network constraints.

Observations, Findings and Verification

Dependency on Number of Hash Functions

A central outcome of both our theoretical analysis (Section 5.4.4) and our experimental evaluations is that **the number of hash functions k is the dominant driver of accuracy**. While our simulations and tables reference "compression rates" (which indirectly determine k), the formal results based on Cohen's MinHash-based estimators and Hoeffding's inequality demonstrate that increasing k reduces the variance and tightens the high-probability bounds on estimation error.

Empirically, we see that for *any* given traffic volume or "raw" data size, accuracy correlates almost entirely with the number of hash functions. Compression can be viewed simply as a mechanism that modifies k. Indeed, low compression rates (such as 4 or 8) may yield larger MinHash signatures, thus allowing more hash functions, whereas higher compression rates (like 128 or 256) reduce k. However, the key takeaway is that *it is the size of k, not the compression factor by itself, that governs accuracy.*

Cluster	**NumOfObjs**	**k**	**Threshold 20%**	**Threshold 40%**	**Threshold 80%**
Accuracy Based on number of hash functions					
Limited	1000	10	79	66	25
Limited	1000	8	82	64	24
Limited	1000	8	75	60	25
Limited	5000	8	83	71	32
Limited	5000	7	82	61	20
Limited	5000	12	81	66	30
Limited	25000	12	78	68	27
Moderate	1000	24	88	71	35
Moderate	5000	28	88	72	44
Moderate	5000	24	81	74	30
Moderate	25000	28	91	80	27
Moderate	25000	32	89	87	39
Moderate	25000	23	81	71	35
High	5000	80	96	94	45
High	5000	112	96	89	45
High	5000	136	95	85	60
High	25000	120	95	91	50
High	25000	96	96	88	51
High	25000	110	95	84	57
Extreme	25000	240	96	96	51
Extreme	25000	284	98	92	57
Extreme	25000	225	97	87	70

Table 6.1: HoMoED Accuracy and Communication Costs reordered by ascending number of hash functions (k).

In our experiments, the accuracy remains high when k is sufficiently large, aligning with the Hoeffding-style bounds. Therefore we can simplify the Table 6.1, and create a more comprehensive and generelized Table 6.2, based on number of hash functions :

Cluster	**NumOfObjs**	**k**	**Threshold 20%**	**Threshold 40%**	**Threshold 80%**
Accuracy Based on number of hash functions					
Limited	1000	5-15	75-85	60-70	25-30
Moderate	1000	15-35	80-90	70-80	30-40
High	5000	60-120	90-95	80-90	45-55
Extreme	25000	150-300	95-99	85-95	50-60

Table 6.2: HoMoED Accuracy and Communication Costs, clustered by number of hash functions (k).

Conversely, when k is small (e.g., fewer than 15 hash functions), we observe rapid degradation in accuracy, especially at higher detection thresholds. For instance:

- With a very low number of hash functions ($k \in [1, 4]$), accuracy for moderate thresholds (e.g., 40%) hovers around 40–50%, dropping to $\leq 25\%$ for high thresholds (e.g., 80%).
- With a limited number of hash functions ($k \in [5, 15]$), accuracy for moderate thresholds (e.g., 40%) hovers around 60–70%, dropping to $\sim 25\%$ for high thresholds (e.g., 80%).
- With a moderate number of hash functions ($k \in [15, 35]$), accuracy improves to the 70–80% range for mid thresholds and remains around 30–40% for very high thresholds.
- With a high number of hash functions ($k \in [60, 120]$), the system maintains considerably higher accuracies across all thresholds.
- With an huge number of hash functions ($k \gg 100$), the system maintains considerably higher accuracies across all thresholds, often exceeding 95%.

Experimental results verify in multiple network scenarios and proves that accuracy *depends on the number of hash functions*, rather than on the raw data size or the nominal compression factor, as Hoeffding's Inequality theoretically proves.

Compression's Role: Comparing Communication Costs.

While accuracy is dictated solely by the number of hash functions k, compression plays a crucial role in optimizing communication costs. By reducing the size of transmitted data, compression can significantly lower network congestion and improve scalability, particularly in high-traffic scenarios. However, overly aggressive compression reduces k, leading to accuracy degradation.

In cases of low traffic, the overhead of hashing and compressing may outweigh its benefits, making raw data transmission more practical. Additionally, in latency-sensitive applications, the computational cost of MinHashing might introduce delays, affecting real-time performance. In summary, while compression does not impact accuracy, it is essential for balancing bandwidth efficiency, computational overhead, and practical feasibility. The decision to compress should be guided by system constraints and network conditions to ensure an optimal trade-off between accuracy and communication efficiency.

Identifying the Compression Turning Point.

Small compression rates at high traffic volumes generate large MinHash signatures, increasing NACK-related overhead. For instance, for 25,000 objects and a compression rate of 4, communication costs reach 16MBytes with 97% accuracy. At a compression rate of 256, costs drop to 10kBytes, but accuracy declines to 65%. This trade-off highlights the non-linear relationship between compression and communication efficiency. The *compression turning point* is more prominent in high-traffic networks but may not appear in low-traffic conditions.

The following plot visualizes gain-factor variations across compression levels, with colors indicating compression rates: blue (4), green (8), orange (16), and purple (32). Additionally, the plot includes parallel dotted lines (y = 4, 8, 16, 32) to represent the compression levels. These lines help identify the points where the communication gain-factor performs worse than the corresponding compression factor. Specifically, if the line graph for a given compression level falls below the corresponding y = X line, the communication gain is less than the compression factor at that point.

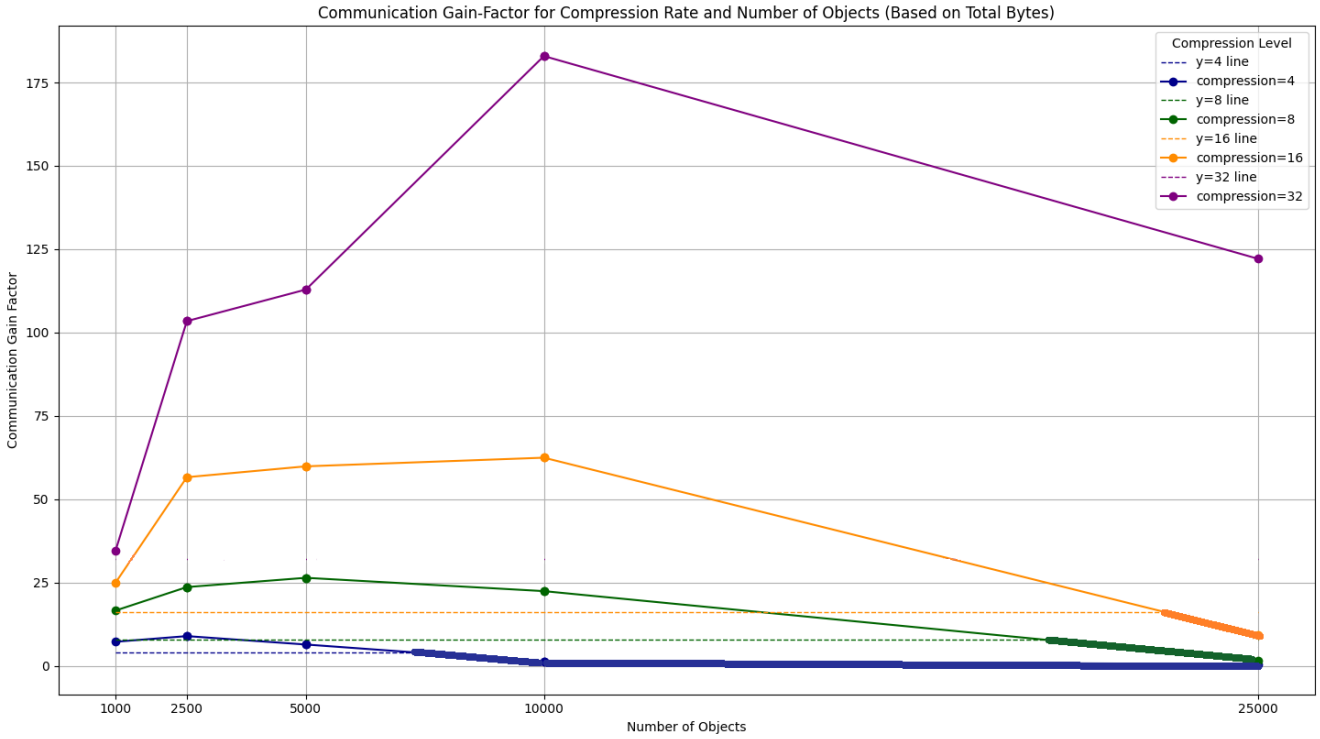


Figure 6.16: Communication Gain-Factor - Compression = 4, 8, 16, 32.

From the plot, we observe that for instance ,for compression = 4, the communication gain-factor falls below the y = 4 line for 10K and 25K objects. For compression = 8 and 16, this occurs for 25K objects. For compression = 32 and higher, the communication gain consistently exceeds the compression factor, indicating improved efficiency at these levels.

Figure 6.16 shows also the existance of an optimal point, where the compression factor achieves the most efficient balance between communication costs and system performance. This peak varies depending on the compression level and traffic volume, representing the conditions under which communication is most optimized. For instance, for a compression factor of 4, the optimal traffic volume appears to be approximately 2,5K objects, suggesting that at this scale, the trade-off between compression and communication overhead is minimized. In contrast, for compression = 32, the peak efficiency is observed at a traffic volume of around 10K objects. Similarly, for compression = 16, the optimal traffic volume shifts to approximately 10K objects, and for compression = 8 the best performance at a traffic volume of 5K objects.

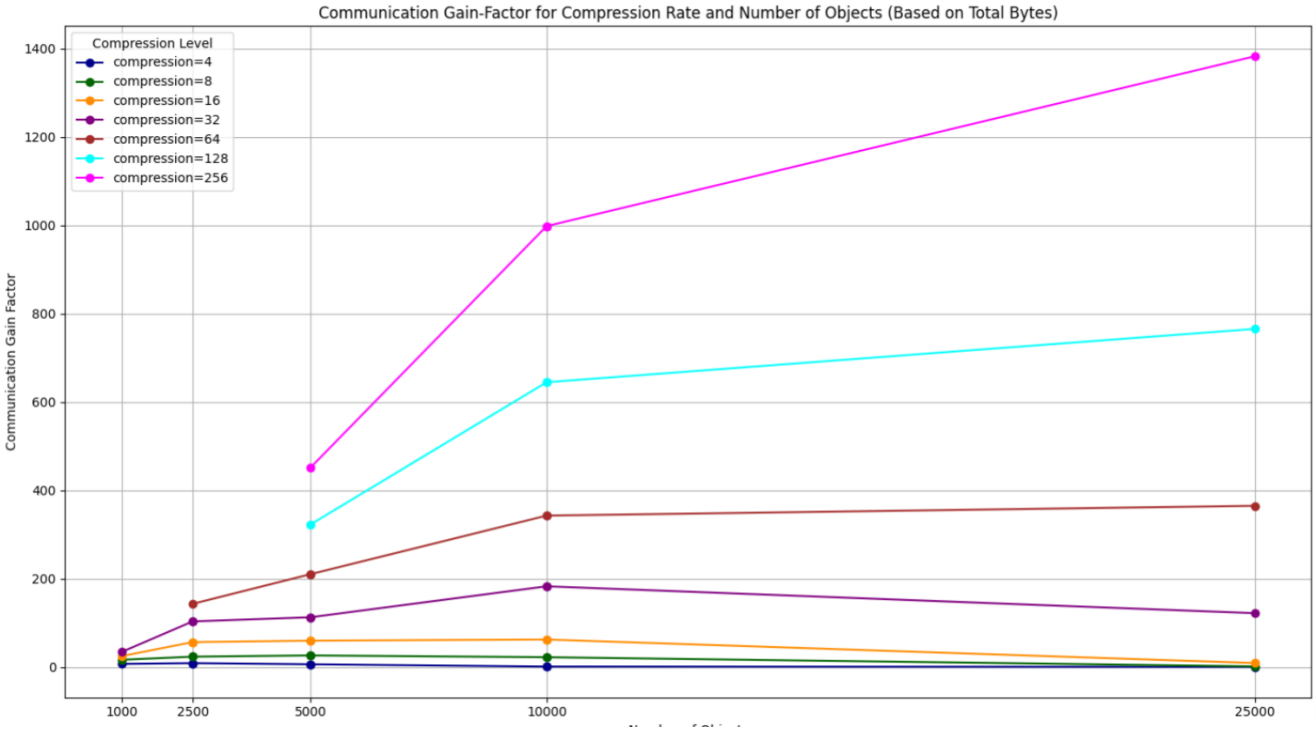


Figure 6.17: Communication Gain-Factor - All Compressions.

These observations highlight the intricate relationship between compression levels and traffic volumes, underscoring the importance of selecting a compression factor that aligns with the expected network conditions. The Figure 6.17 extends this analysis by incorporating all compression levels, offering a comprehensive view of the behavior of communication gains as a function of compression. This visualization clearly reveals the existence of a peak, or best point, for each compression level, further emphasizing the importance of tuning compression strategies to maximize network performance.

6.2.4 HoMoEdges Detection regarding Traffic Volume

Now, we will reveal the impact of traffic volume on accuracy by analyzing our accuracy results from a different perspective, which better reveals and visualizes the effect of traffic volume :

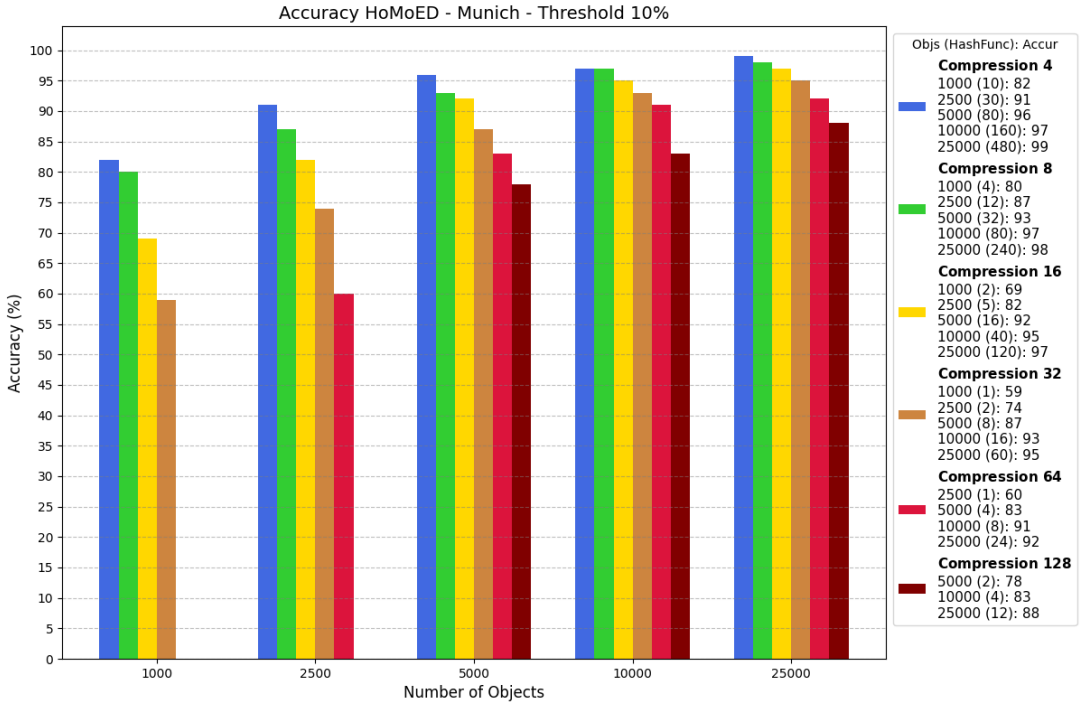


Figure 6.18: Munich network, 180 sensors, 10% Threshold.

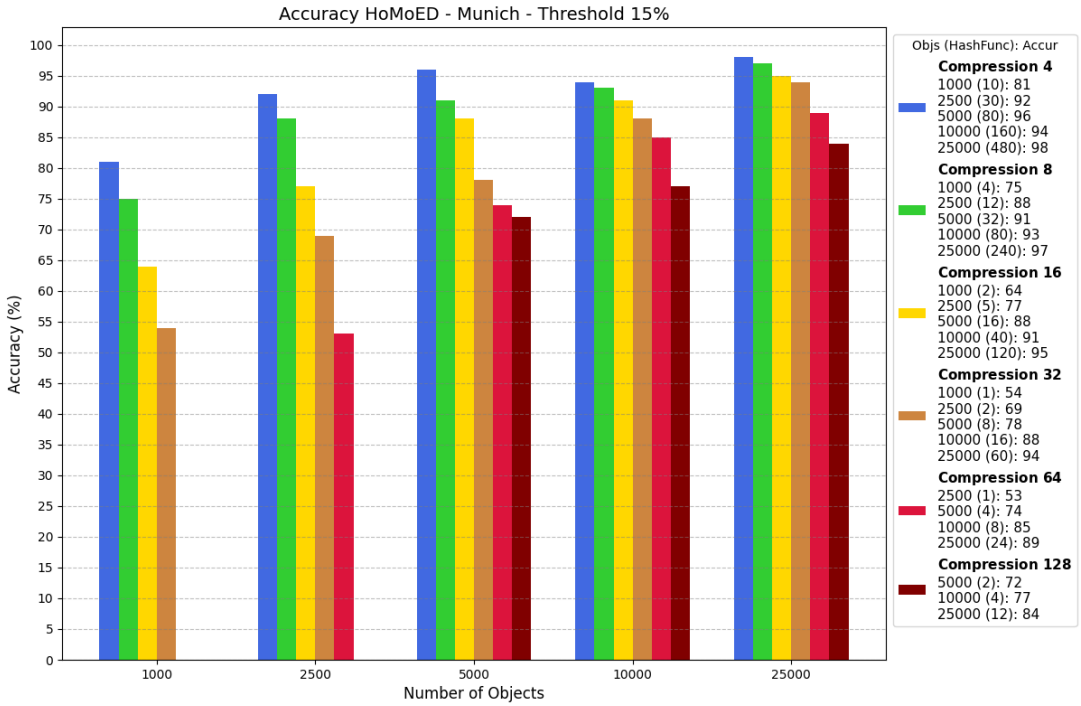


Figure 6.19: Munich network, 180 sensors, 15% Threshold.

These plots reveal the impact of an increased compression rate (corresponding to an increased number of hash functions), which improves precision as the traffic volume increases. In the legend, each compression rate is represented as: *number of objects (number of hash functions): accuracy%*.

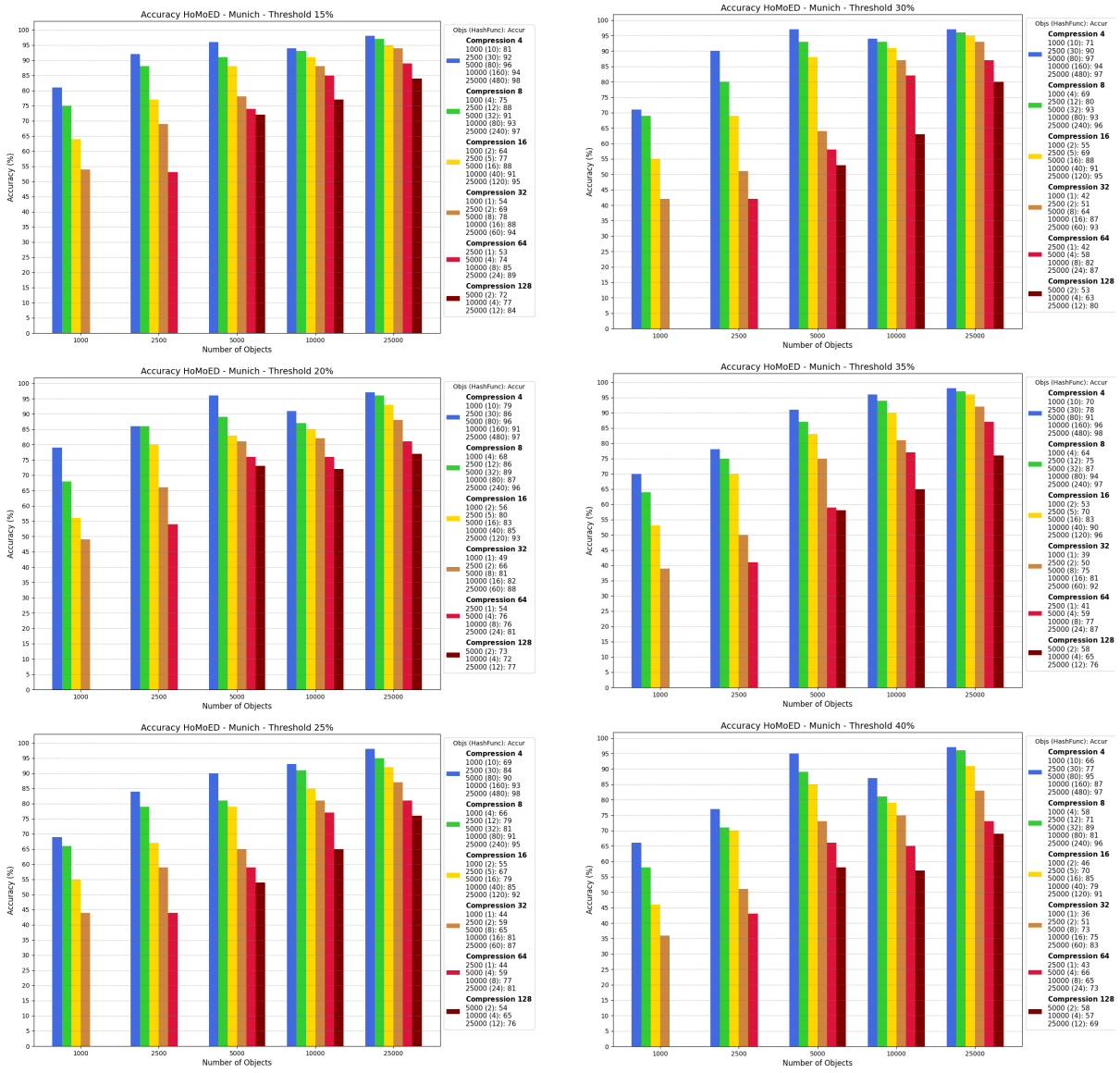


Figure 6.20: Impact of Traffic Volume on HoMoEdge Detection Efficiency for Thresholds between 15% and 40%

Increasing compression leads to a loss of accuracy. However, as we raise traffic volume, the use of additional hash functions enhances accuracy across all compression levels. From this perspective, it is clear that achieving acceptable accuracy at higher thresholds necessitates the implementation of more hash functions. For instance, at a threshold of 10%, accuracy remains relatively high across various traffic volumes. Conversely, at a threshold of 80%, maintaining good accuracy is only feasible with higher traffic volumes or by utilizing a greater number of hash functions.

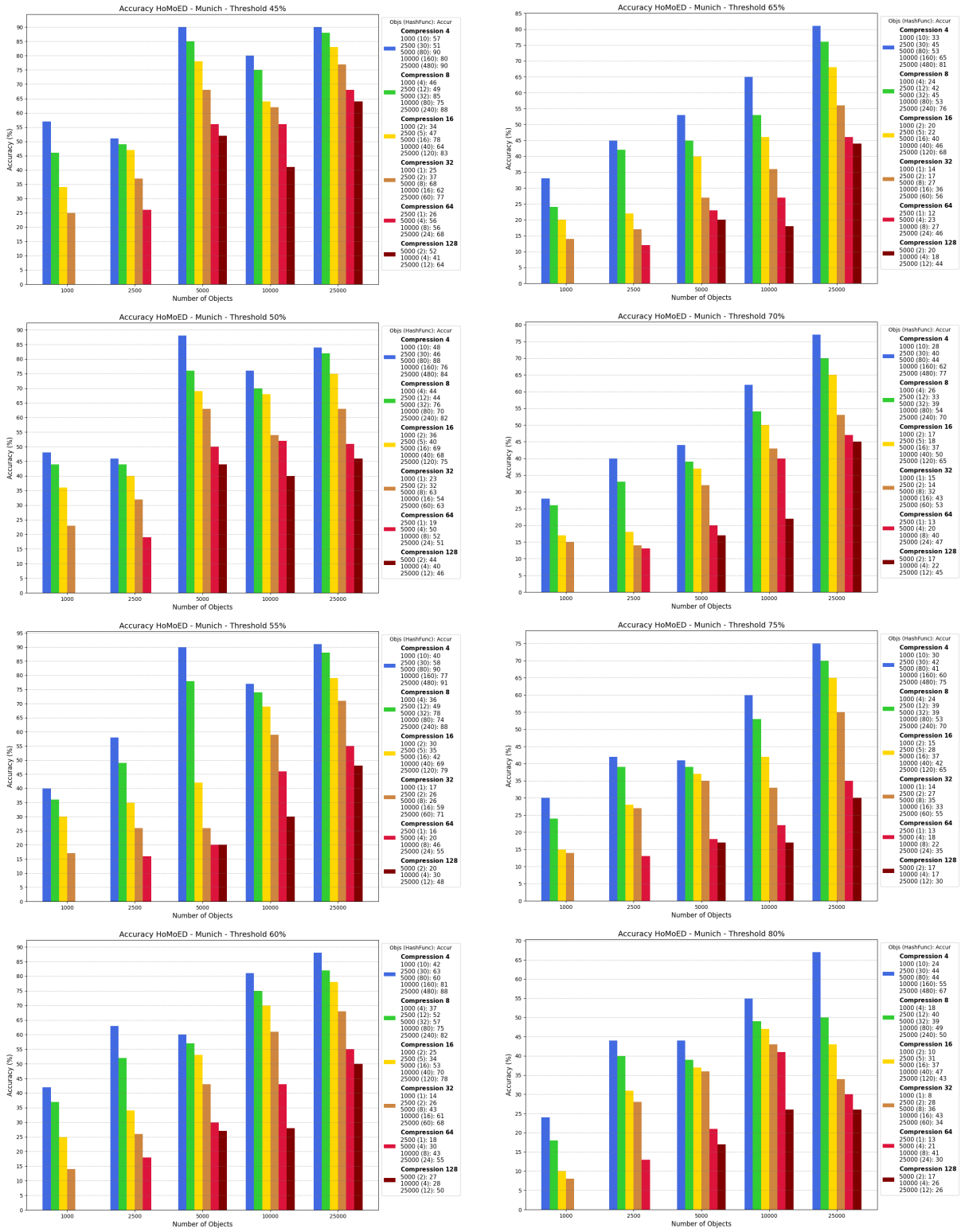


Figure 6.21: Impact of Traffic Volume on HoMoEdge Detection Efficiency for Thresholds between 45% and 80%

6.2.5 HoMoEdges Detection regarding Threshold

Here provide an overview and summarize how accuracy evolves as the threshold dynamically changes, and how it influences each compression level.

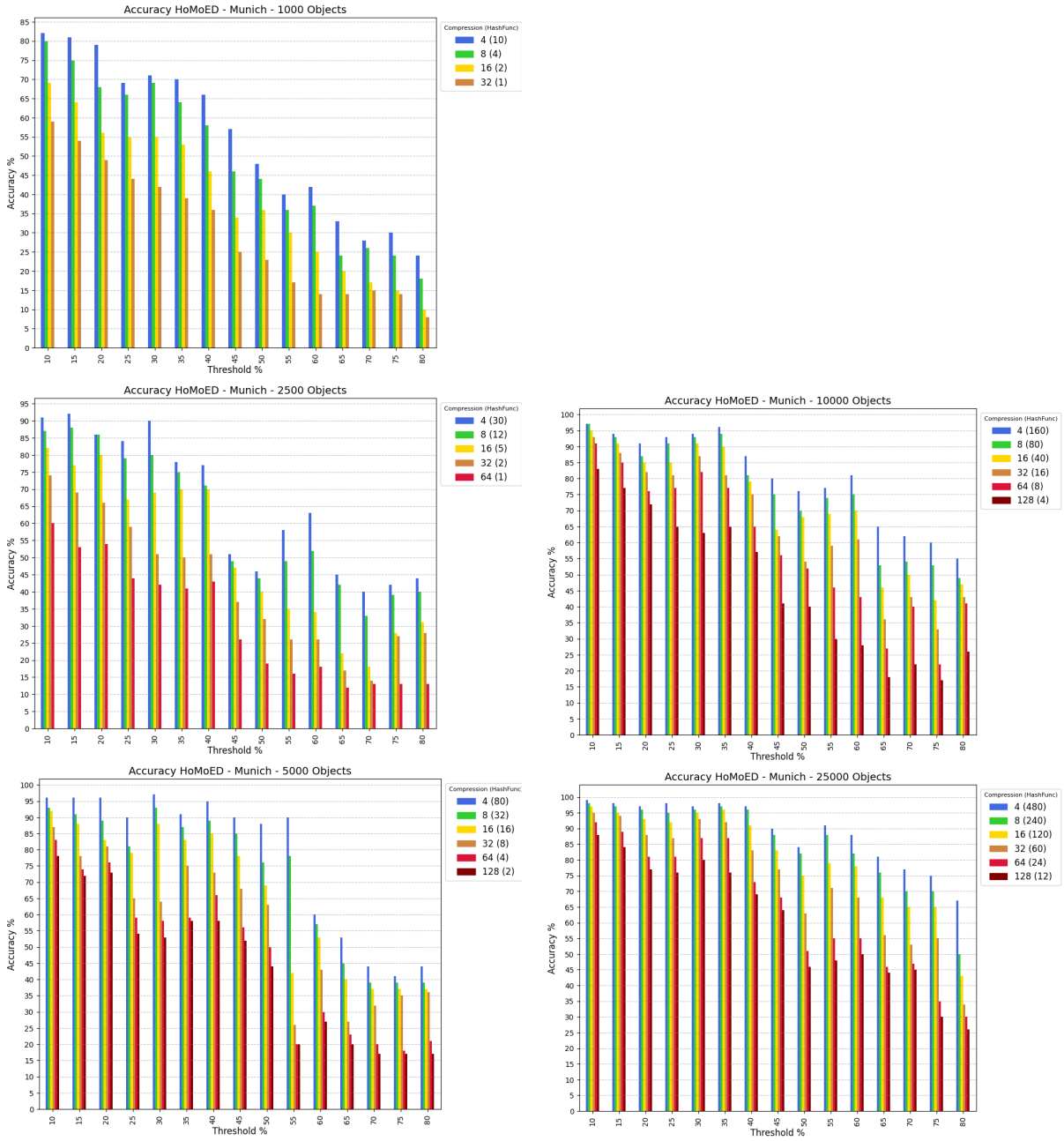


Figure 6.22: Impact of Threshold on HoMoEdge Detection Efficiency for different Traffic Volumes

It is clearly visible that by increasing the threshold , we are loosing accuracy. We can observe this relationship for all the compression levels and different ammounts of hash functions, retaining always the accuracy of lower compression(more hash functions) on a higher level than the accuracy of higher compression(less hash functions).

6.2.6 Effect of Hash Functions

All prior results were obtained by repeating the same experiment multiple times using distinct sets of hash functions. In real scenarios, the same experiment may sometimes deviate from ideal behavior, showing certain anomalies or variations. To illustrate this, we present an example comparing the results of a single experiment conducted with one random seed (one set of hash functions) and the results after 2 and five repetitions with different random seeds.

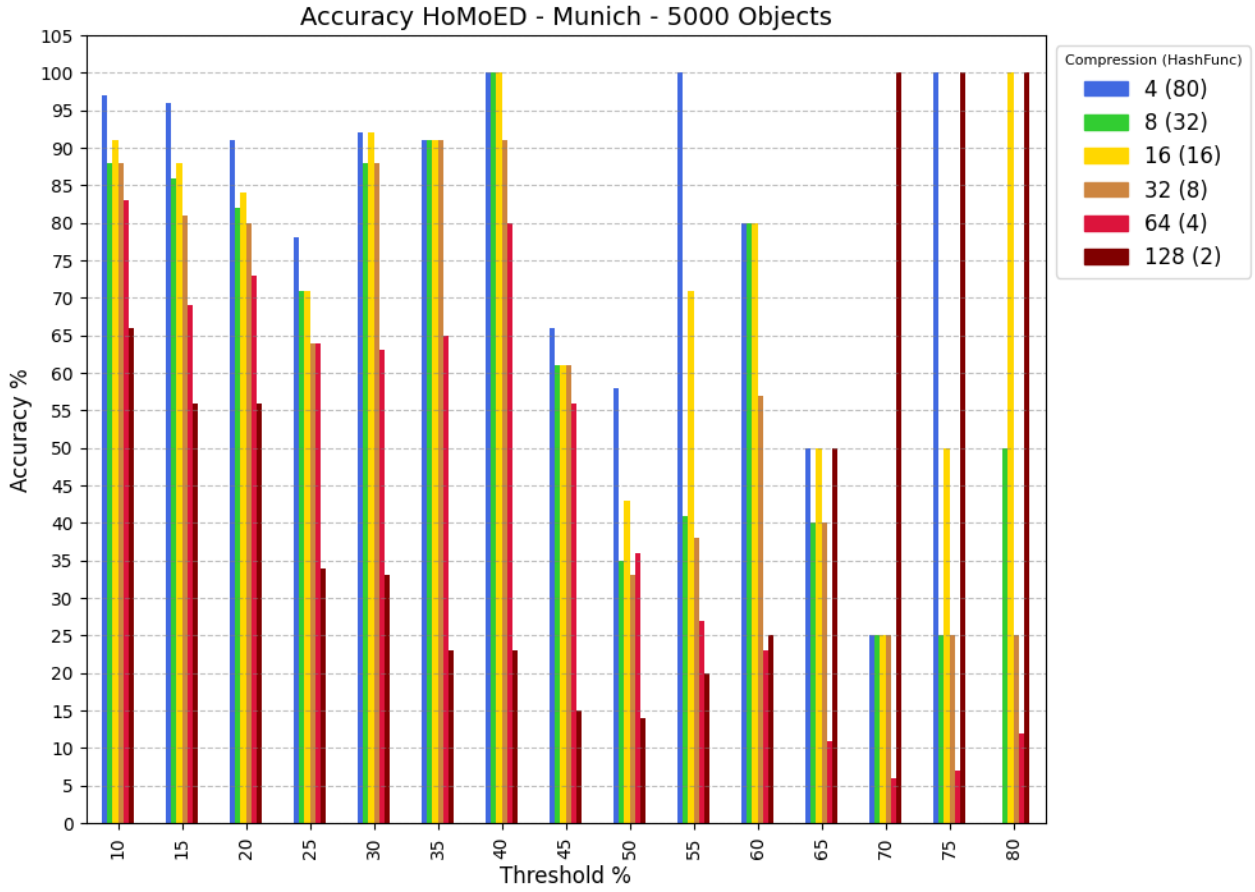


Figure 6.23: Result of One Experiment with a Single Random Seed

We observe that the accuracy does not consistently align with the expectations derived from the compression rates, due to the random nature of the hash functions employed. Notably, lower compression yields better results compared to higher compression levels. However, as the threshold increases, the resulting behavior becomes increasingly erratic. It is essential to emphasize that this unpredictability is closely tied to the number of active motion paths corresponding to each threshold. Using the HoMoEdgeDetection algorithm (see Algorithm 5), even when the predicted size of the union of all objects is almost accurate, edge cases can arise—such as the occurrence of a single homoedge. A prediction that slightly exceeds the actual size could mistakenly classify this homoedge as a false negative. On the other hand, while a lower prediction may result in a more significant overall accuracy error, it has the advantage of capturing the homoedge.

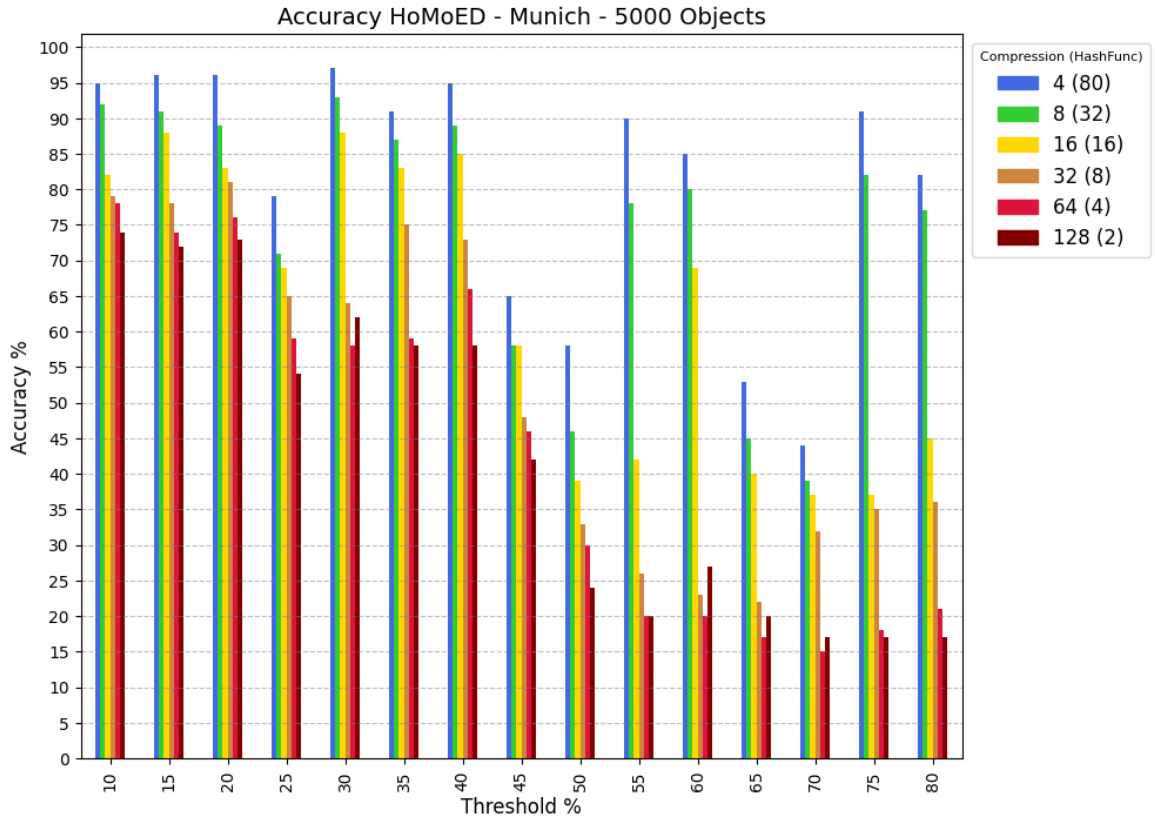


Figure 6.24: Result After 2 Repetitions with Different Random Seeds

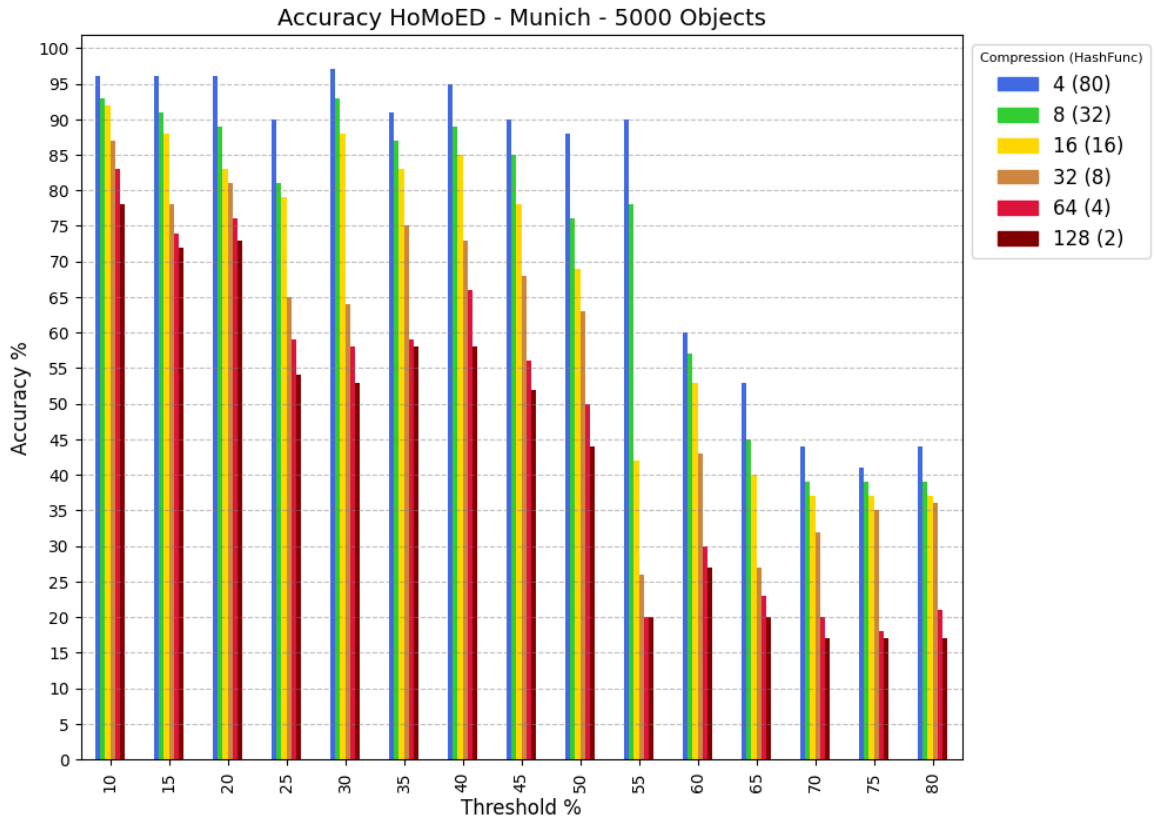


Figure 6.25: Result After 5 Repetitions with Different Random Seeds.

As demonstrated, conducting multiple experiments with various hash functions enhances the consistency and reliability of the results. With an increased number of repetitions, the diagrams tend to become smoother and more stable. These findings underscore the importance of acknowledging the uncertainty associated with a single experiment that utilizes random hash functions.

6.2.7 Conclusion - Scientific Implications

Our results highlight the intricate relationship between traffic volume, compression ratios, thresholds, and accuracy in the *HoMoED* algorithm. We observe trade-offs and patterns that influence both performance and resource efficiency, depending on the chosen parameters. From these observations, we derive several key insights and scientific implications:

- **Number of Hash Functions, Accuracy and Communication Costs:** Accuracy is dictated by the number of hash functions (k), not the compression factor. A new route ,for the already established error bound by Cohen [20], derived from Hoeffding's inequality, confirms that accuracy is independent of the raw set length or compression rate and solely depends on k. This also serves as the primary factor influencing communication costs, as an objective number of hash functions provides a concrete basis for evaluating these costs.

- **Threshold Sensitivity and Fine-Tuning:** Accuracy declines more sharply with higher thresholds and fewer hash functions, emphasizing the need for careful tuning of the *minHoMoSup* parameter. Lower thresholds improve accuracy but increase overhead, while higher thresholds reduce overhead at the risk of missing critical edges.

- **Impact of Randomization in Hash Functions:** The algorithm's performance varies due to the randomness in hash function selection, leading to inconsistencies, particularly in smaller datasets or under heavy compression. Exploring deterministic or adaptive hashing methods could improve reliability and robustness.

- **Compression Turning Points and Non-linear Behavior:** We observe a critical *compression turning point*, particularly in high-traffic scenarios with lower compression rates, where the communication gain begins to exceed the compression ratio. Below this point, gains are worse than the compression factor, and in some cases, communication costs increase due to raw data transmission overhead. Above the turning point, the communication gain follows a non-linear positive progression, eventually reaching a peak. Beyond this peak, while gains continue, the rate of improvement diminishes. However, the gains always remain better, or significantly better, than the compression factor, revealing an optimal balance between compression rate and traffic volume.

6.3 Munich Network - HoMoPaD

Building upon our earlier work, we now focus on detecting Hot Motion Paths (HoMoPaD) to validate the mathematical formulations. Our analysis is based on evaluating the accuracy between raw and min-hashed results across all generated paths, regardless of their length. By performing a transitive closure, an algorithm capable of detecting HoMoEdges can inherently identify extended HoMoPaths. By performing a transitive closure, an algorithm capable of detecting HoMoEdges inherently can identify extended HoMoPaths. We divide our evaluation into two parts:

1. **Realistic Scenario with Errors in HoMoEdge Detection:** HoMoPath detection including the accuracy impact of G-LESE-related errors.
2. **Ideal Scenario without Errors in HoMoEdge Detection:** HoMoPath detection under optimal conditions where G-LESE detects all HoMoEdges flawlessly.

6.3.1 HoMoPaD : Ideal Scenario without HoMoED-Errors

In this section, we evaluate our algorithm's performance in detecting HoMoPaths under ideal conditions. We consider a scenario where the HoMoEdge detection phase is flawless, resulting in no false positives or negatives. By isolating the HoMoPaths detection phase from inaccuracies in the HoMoEdge process, we analyze the HoMoPaD algorithm's core ability to identify HoMoPaths accurately. This approach provides insight into its potential when unaffected by earlier detection errors.

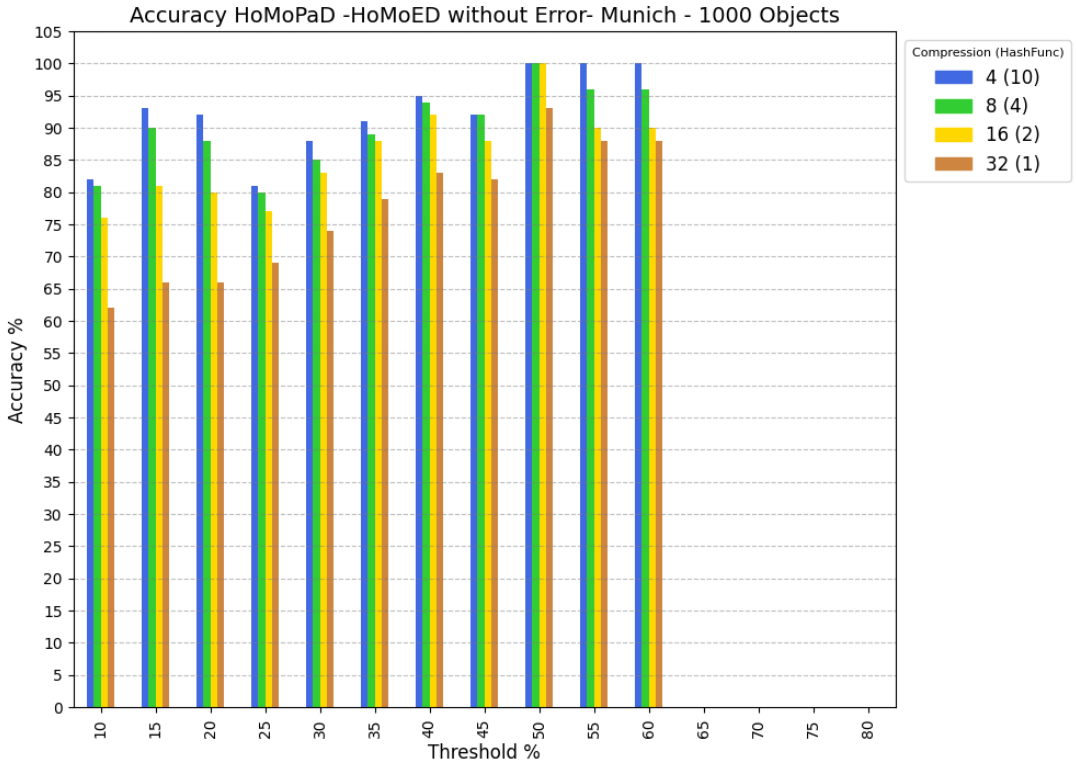


Figure 6.26: HoMoPaD (Ideal Scenario), 180 sensors, 1000 tagIDs.

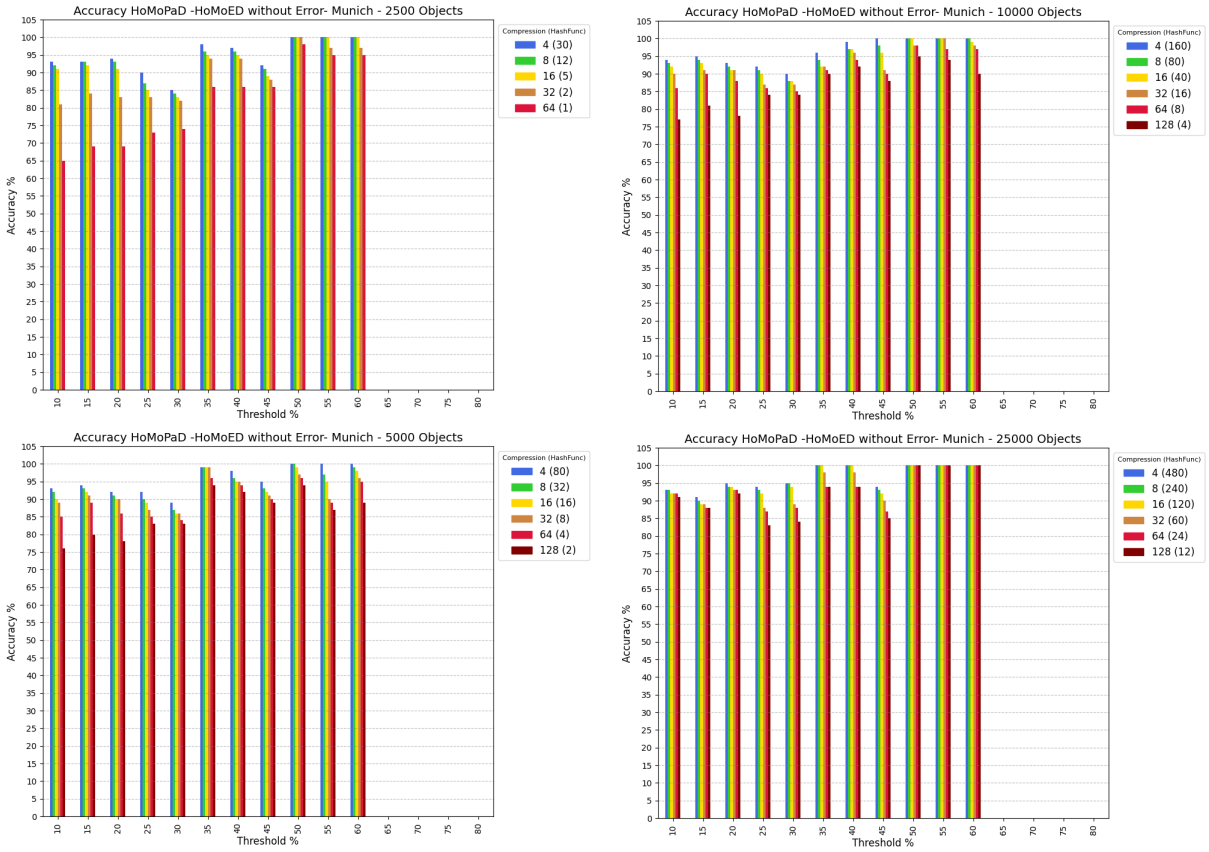


Figure 6.27: HoMoPaD (Ideal Scenario), 180 sensors, 2,5k ...25k tagIDs.

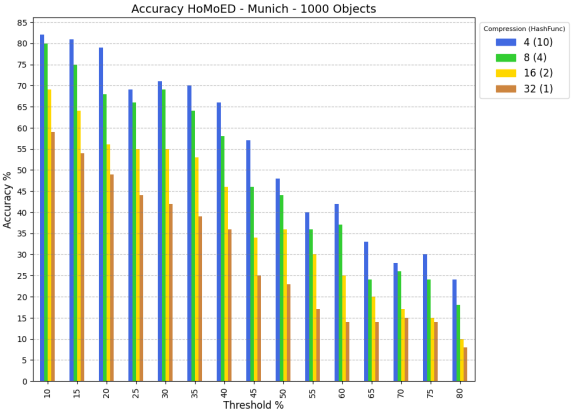
The experimental results in Figures 6.27 confirm the predictions from Section 5.3.1. HoMoPaD classifies paths as hot based on the ratio of the intersection of involved edges in a path to the union of all edges in the region, as defined in Equation (4.4). Since MinHash approximates this ratio via Jaccard similarity, small deviations from raw-data calculations were expected. At lower thresholds, the denominator includes almost all edges in the region, introducing noise. This effect is more pronounced in large networks where less relevant edges inflate the denominator, making the MinHash-based approximation less precise.

As the threshold increases, the denominator shifts toward high-traffic edges, reducing noise and improving accuracy. Additionally, the first-level communication protocol inherently limits the expansion of the universe set in each distributed region. As a result, each distributed region remains relatively small, preventing excessive data collisions and ensuring that the union of edges in the region remains manageable, further improving the accuracy of the MinHash-based approximation.

In conclusion, accuracy improves at higher thresholds as the denominator focuses on high-traffic edges. The first-level communication protocol naturally limits the size of the universe set, reinforcing this improvement. These results validate that HoMoPaD effectively approximates the global hot-path criterion using MinHash-based Jaccard similarity, making it well-suited for distributed road networks.

6.3.2 HoMoPaD : Realistic Scenario with Errors

Now we present the results based on our algorithm's capability to detect HoMoPaths when the HoMoEdge detection is affected by various errors introduced during that phase: Moving Objects = 1000



(a) Accuracy HoMoED

(b) Accuracy HoMoPaD Ideal Scenario

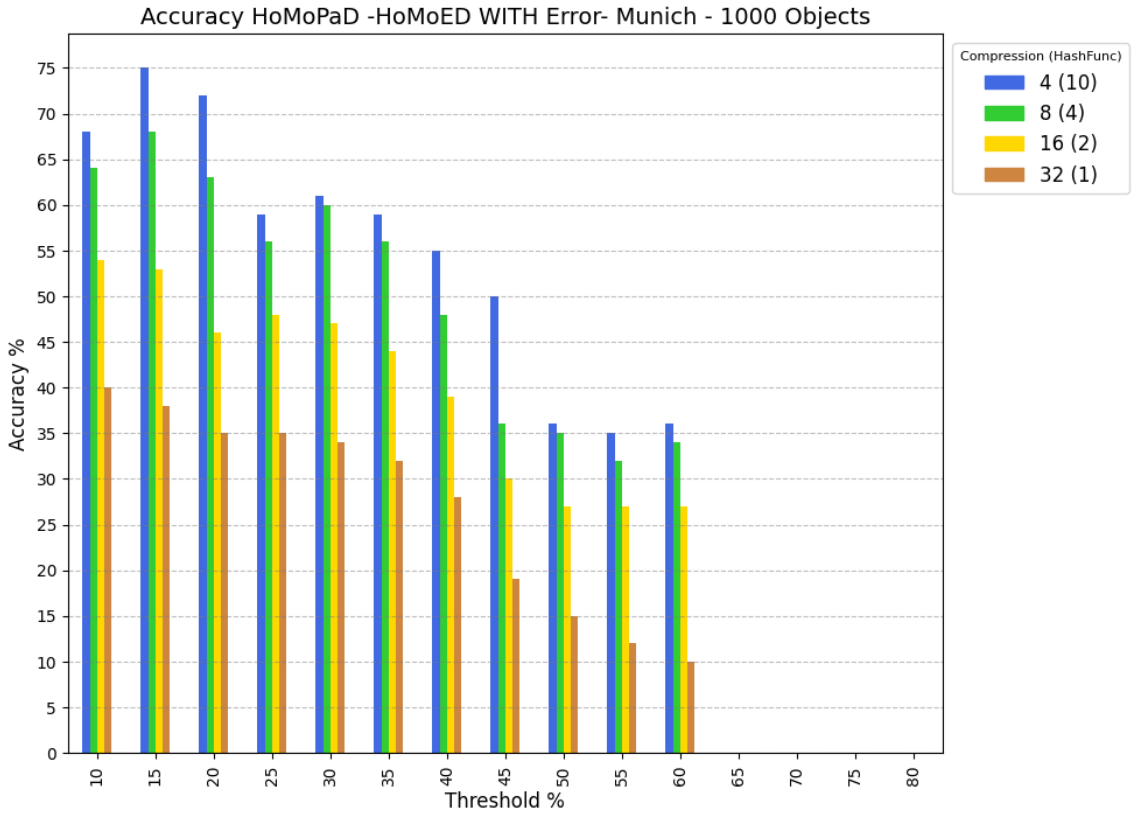
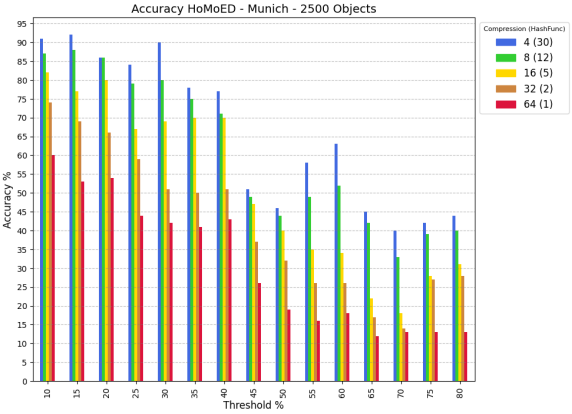


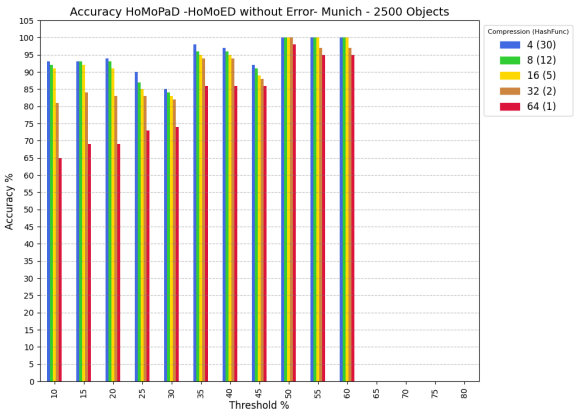
Figure 6.29: Accuracy HoMoPaD Real Scenario

The side-by-side comparison reveals that We are achieving a balance for low thresholds, due to high performance of HoMoED and the lower performance of HoMoPaD, As threshokd increases, HoMoPaD does not degrade the decreased HoMoED accuracy, but it ensures accuracy among the hot motion edges which may create hot motion paths. .

Moving Objects = 2500



(a) Accuracy HoMoED



(b) Accuracy HoMoPaD Ideal Scenario

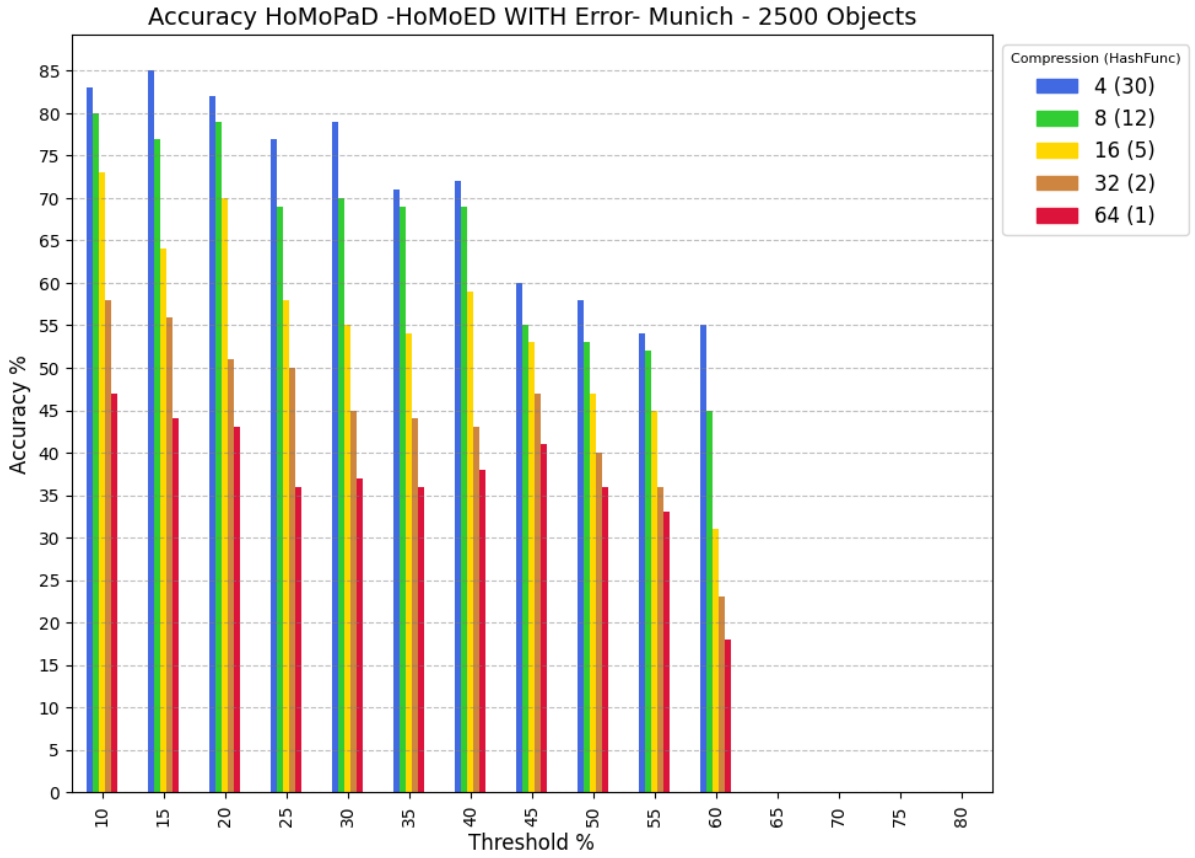
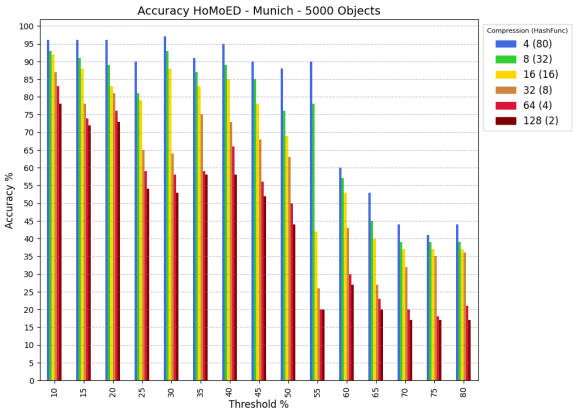


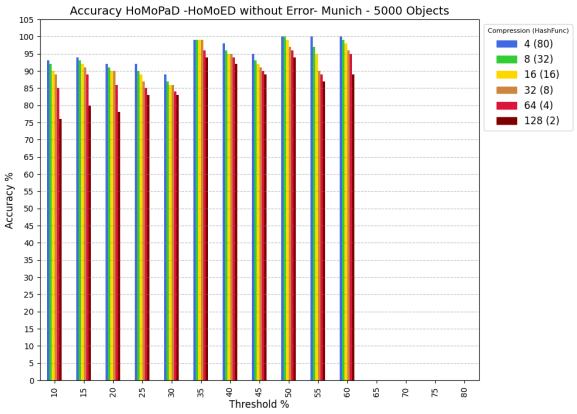
Figure 6.31: Accuracy HoMoPaD Real Scenario

The same trend is observed with higher traffic volume. At lower thresholds (10–20%), HoMoED achieves nearly 90% accuracy for small compressions, while the lower performance of HoMoPaD (ideal scenario) causes a slight decrease in real-scenario accuracy. However, at higher thresholds (e.g., 45%), where compression 4 (30 hash functions) results in HoMoED accuracy below 52%, the ideal-scenario HoMoPaD reaches 92%, allowing the real-scenario HoMoPaD to maintain 60% accuracy.

Moving Objects = 5000



(a) Accuracy HoMoED



(b) Accuracy HoMoPaD Ideal Scenario

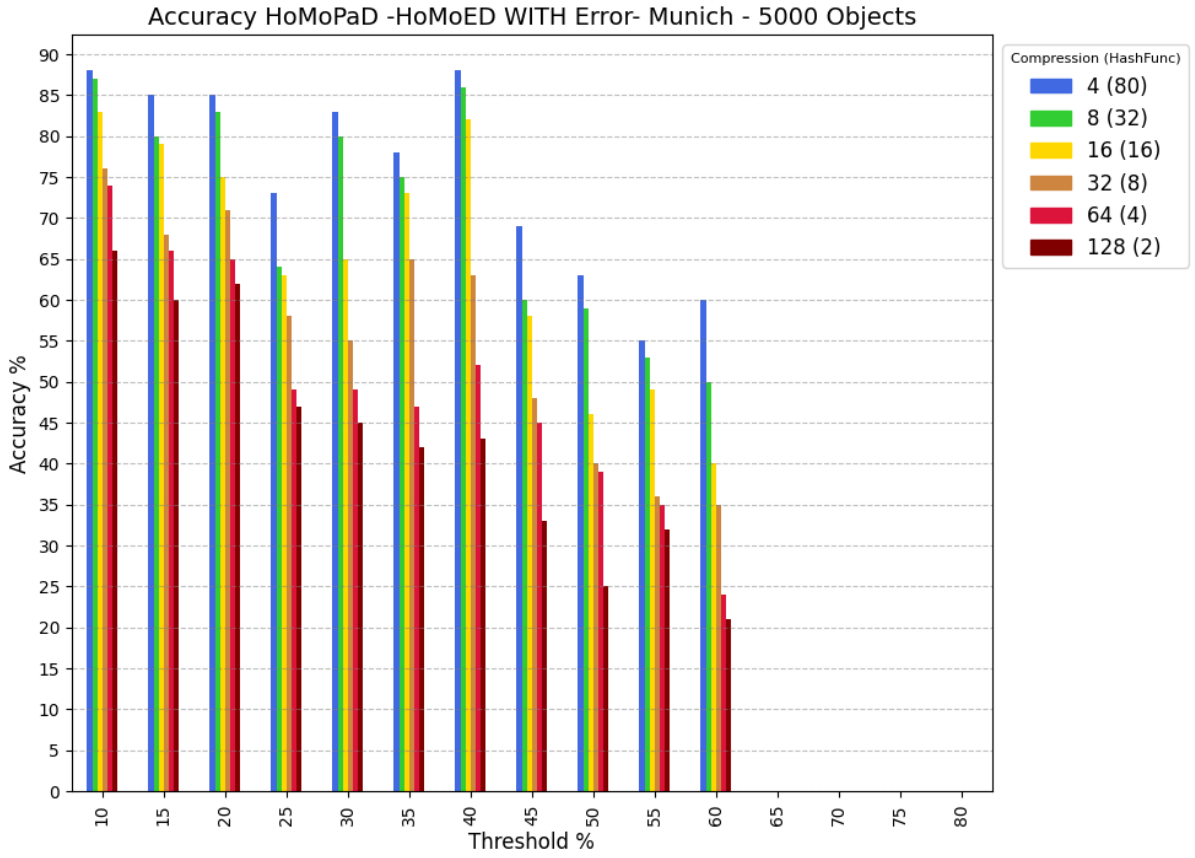
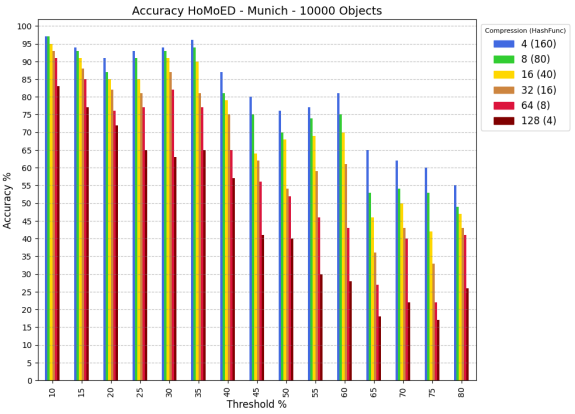


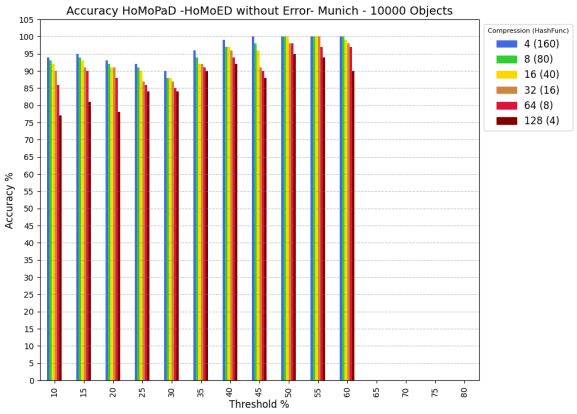
Figure 6.33: Accuracy HoMoPaD Real Scenario

At lower thresholds, HoMoPaD in the real scenario shows a further decrease in accuracy due to the growing impact of HoMoED errors. However, at higher thresholds, the trend stabilizes. For example, at 40% threshold with 32 hash functions, HoMoED achieves 88% accuracy, while HoMoPaD in the real scenario closely follows with 86%, demonstrating its ability to maintain reliable detection despite initial degradations.

Moving Objects = 10,000



(a) Accuracy HoMoED



(b) Accuracy HoMoPaD Ideal Scenario

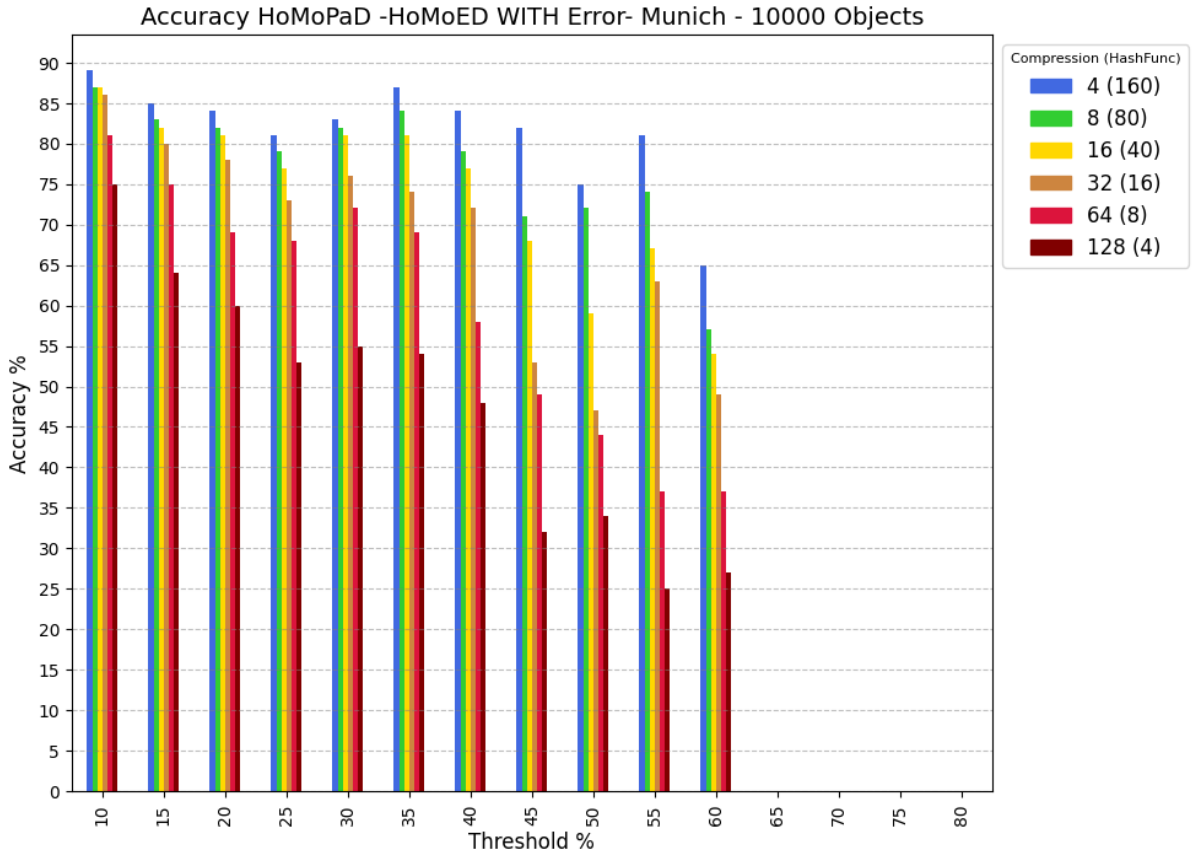
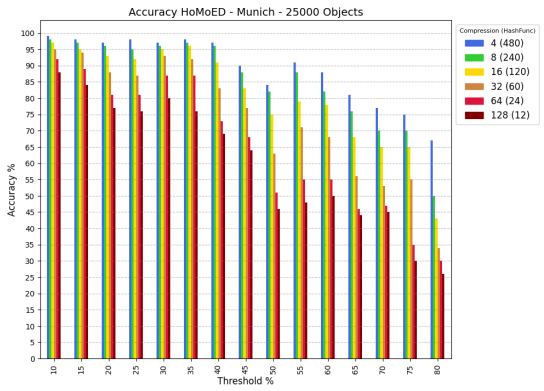


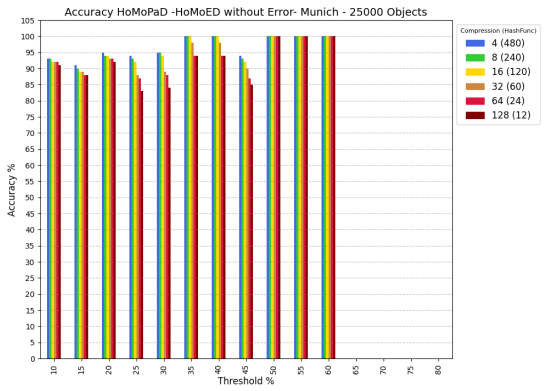
Figure 6.35: Accuracy HoMoPaD Real Scenario

For 10,000 objects, we observe that with a moderate number of hash functions, HoMoPaD ,in the real scenario and achieves accuracy comparable to or even higher than HoMoED, for higher thresholds. For example for 80 hash functions the HoMoED remains under 70% accuracy for thresholds higher than 4ß% , but HoMoPaD achieves accuracy higher than 70%.

Moving Objects = 25,000



(a) Accuracy HoMoED



(b) Accuracy HoMoPaD Ideal Scenario

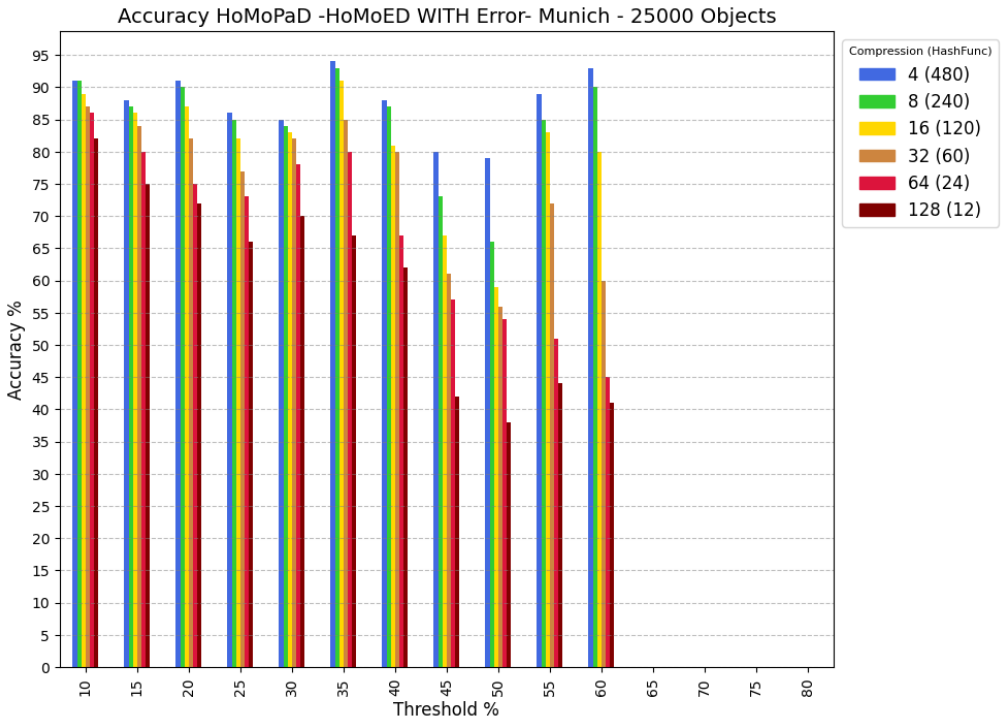


Figure 6.37: Accuracy HoMoPaD Real Scenario

The same trend holds for 25,000 moving objects, particularly for high hash function counts. However, using 480 hash functions is impractical and not necessary as previously discussed. Even for more feasible cases (60–120 hash functions), the same behavior emerges. This further supports the transitive closure principle: HoMoED underperforms when the estimated union in the denominator of Equation (4.3) exceeds the actual union of all edges, while the numerator remains accurate. This overestimation misclassifies certain edges as hot, but those that persist are more likely to form valid paths due to structural road constraints. Thus, despite HoMoED's limitations in edge detection, HoMoPaD effectively reconstructs hot motion paths by leveraging the connectivity of high-confidence edges. The following table summarizes the results:

NumOfObjs	Compr	K	20% Threshold			40% Threshold			60% Threshold		
			homopad Idea	HoMoED	HoMoPaD	homopad Ideal	HoMoED	HoMoPaD	homopad Ideal	HoMoED	HoMoPaD
1000	4	10	92	79	73	95	66	55	100	42	36
1000	8	4	88	67	64	94	58	47	96	36	34
1000	16	2	80	56	46	91	46	39	90	25	27
1000	32	1	64	49	35	83	36	27	88	18	10
10000	4	160	94	91	84	98	88	84	100	80	65
10000	8	80	93	87	82	97	81	78	100	75	57
10000	16	40	92	85	81	97	78	76	99	70	54
10000	32	20	91	82	77	96	75	72	98	62	49
10000	64	9	87	76	68	94	65	57	97	46	37
10000	128	4	77	72	59	92	56	47	90	28	27
25000	4	480	95	97	91	100	97	87	100	87	93
25000	8	240	94	96	90	100	96	86	100	83	90
25000	16	120	94	93	87	100	91	82	100	78	80
25000	32	60	93	88	86	97	83	80	100	67	60
25000	64	24	93	81	75	94	74	67	100	55	45
25000	128	12	92	77	73	94	68	63	100	50	41

Table 6.3: HoMoED-PaD Accuracy for Different Thresholds

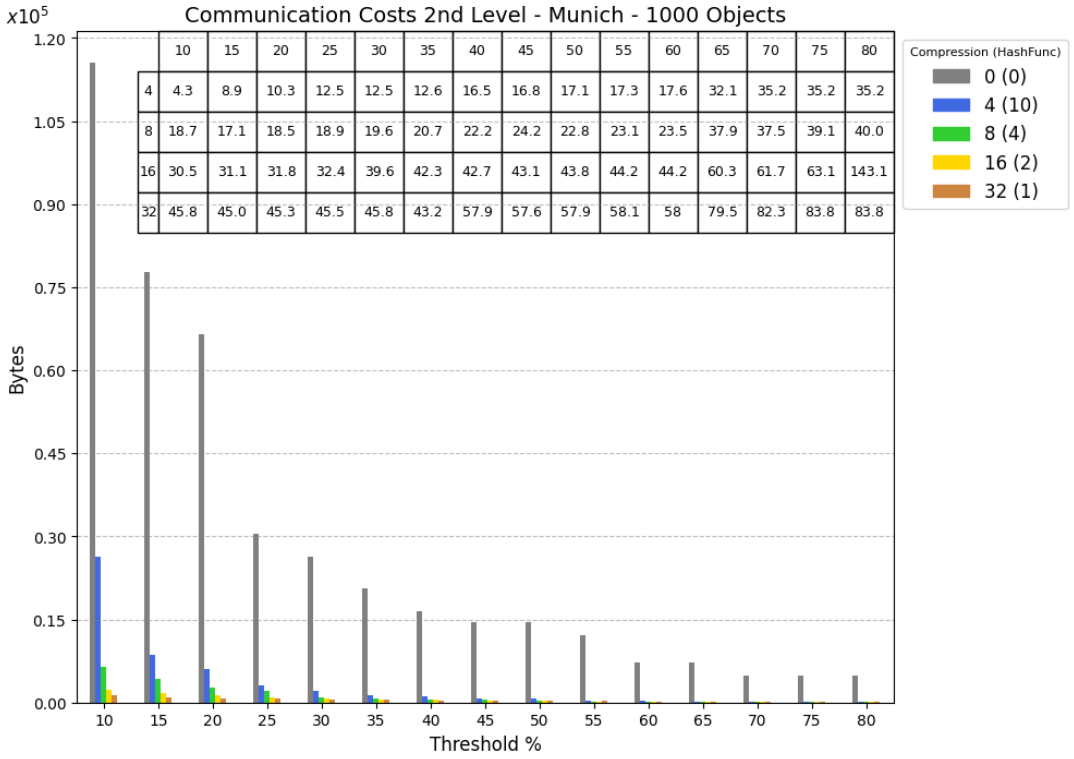
The results confirm that HoMoPath detection closely follows HoMoEdge detection. The table summarizes accuracy values across different moving object counts, compression levels, and hash functions (K) for three threshold levels: 20%, 40%, and 60%. It allows us to directly compare HoMoPaD under ideal conditions (without HoMoED errors), HoMoPaD in a real scenario, and HoMoED itself.

As expected, lower thresholds amplify the gap between HoMoPaD (ideal) and HoMoPaD (real) due to error propagation from HoMoED. However, as the threshold increases, this gap narrows, supporting our theoretical predictions. At the 60% threshold with 120 hash functions, HoMoPaD (real) maintains 80% accuracy despite HoMoED's accuracy being 78%. The trend remains consistent even for 25,000 moving objects, reinforcing that increasing the threshold reduces the impact of early-stage inaccuracies.

Additionally, in several cases, HoMoPaD (real) performs well despite moderate HoMoED errors, suggesting a possible link to the transitive closure principle. When HoMoED overestimates the union in Equation (4.3), weaker edges are misclassified, but those that persist are more likely to form valid paths due to structural road constraints. This empirical observation indicates that, even when HoMoED struggles with individual edges, HoMoPaD can still reconstruct meaningful motion paths by leveraging high-confidence edges. While it cannot fully compensate for severe HoMoED errors, its accuracy stabilizes at higher thresholds, where weak edges are naturally filtered out.

6.3.3 Communication Costs 2nd Level - Sequential Query Processing (SQP) - No COllisions -

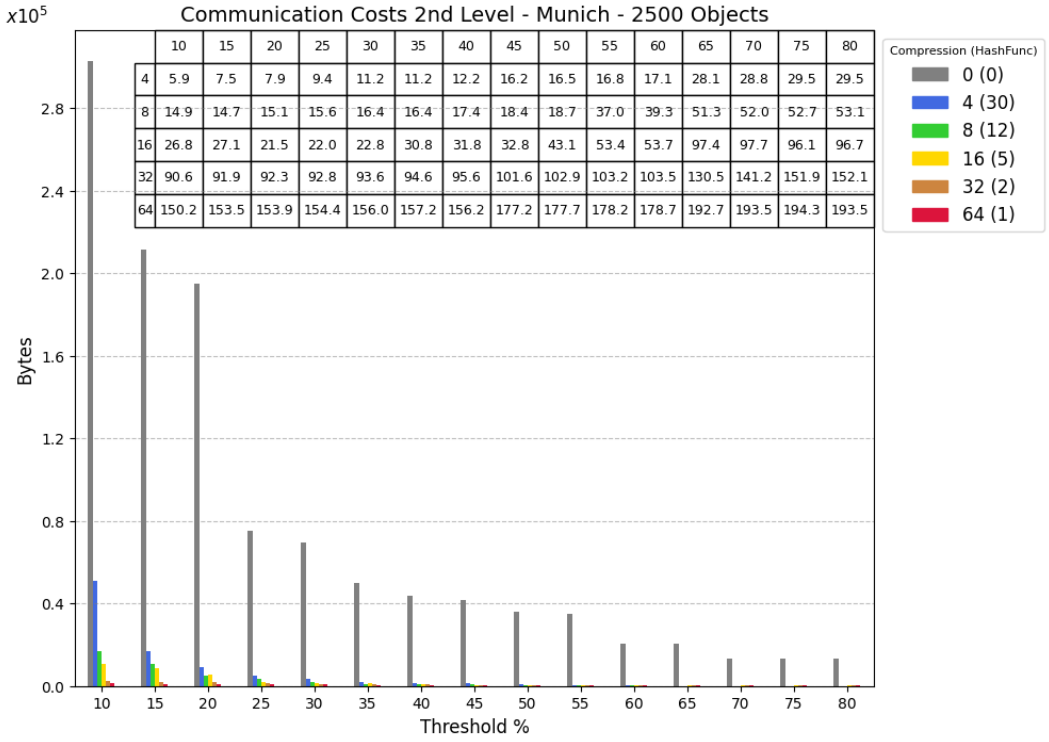
Our analysis extends to calculating the associated communication costs resulting from varying thresholds in the second level. The communication is structured using JSON payloads, as outlined in subsection 3.3.3. It is important to remind that our primary focus does not lie in the communication costs at the second level; this level predominantly reflects individual behaviors and states of traffic, which are inherently unpredictable and do not yield consistent scientific observations. Nevertheless, we will provide visual representations to illustrate the impact of second-level communication costs on the overall communication expenses for individual cases.



	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80
4	4.3	8.9	10.3	12.5	12.5	12.6	16.5	16.8	17.1	17.3	17.6	32.1	35.2	35.2	35.2
8	18.7	17.1	18.5	18.9	19.6	20.7	22.2	24.2	22.8	23.1	23.5	37.9	37.5	39.1	40.0
16	30.5	31.1	31.8	32.4	39.6	42.3	42.7	43.1	43.8	44.2	44.2	60.3	61.7	63.1	143.1
32	45.8	45.0	45.3	45.5	45.8	43.2	57.9	57.6	57.9	58.1	58	79.5	82.3	83.8	83.8

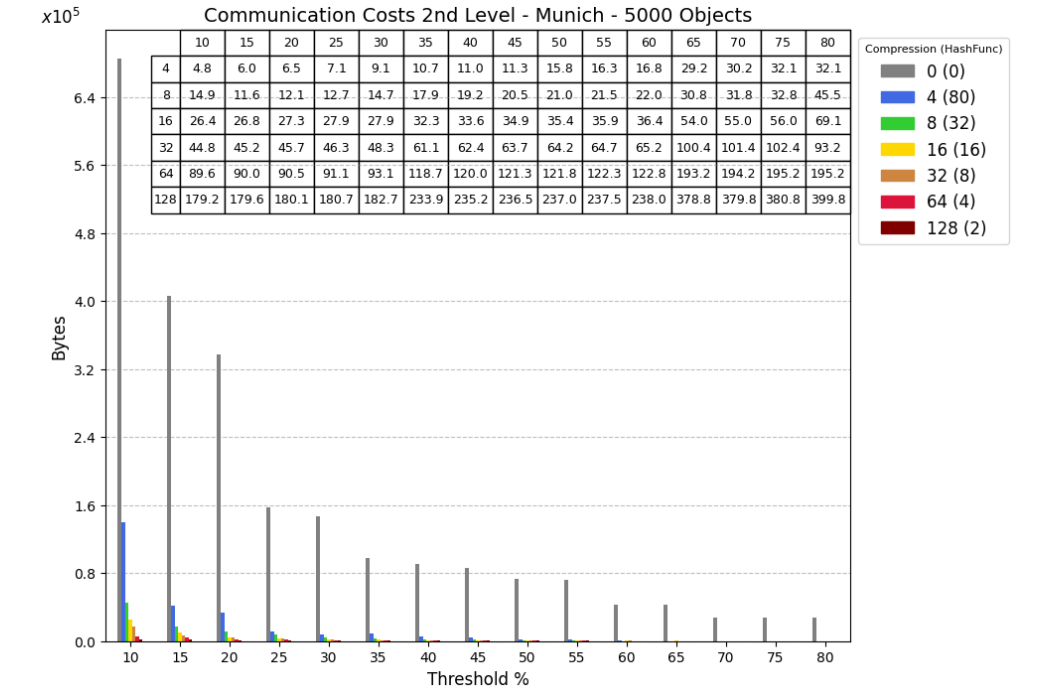
Figure 6.38: Communication costs 2nd Level - Munich network, 180 sensors, 1000 tagIDs.

At this level, we do not distinguish best, worst, or average communication cases due to the absence of collisions; the Top Leader queries Regional Leaders sequentially (head-to-tail). A clear relationship between compression factor and communication cost reduction remains. At lower thresholds, many paths are marked hot, increasing comparisons and false positives in MinHashed mode, which adds extra communication. As thresholds increase, fewer false positives occur, and gains from compression become more stable. However, slight deviations may still arise due to randomness in hash functions, especially with fewer hash functions, affecting detection accuracy. Hence, higher compression does not always guarantee lower cost, as false positives or missed overlaps can offset gains—emphasizing the need to balance compression and accuracy.



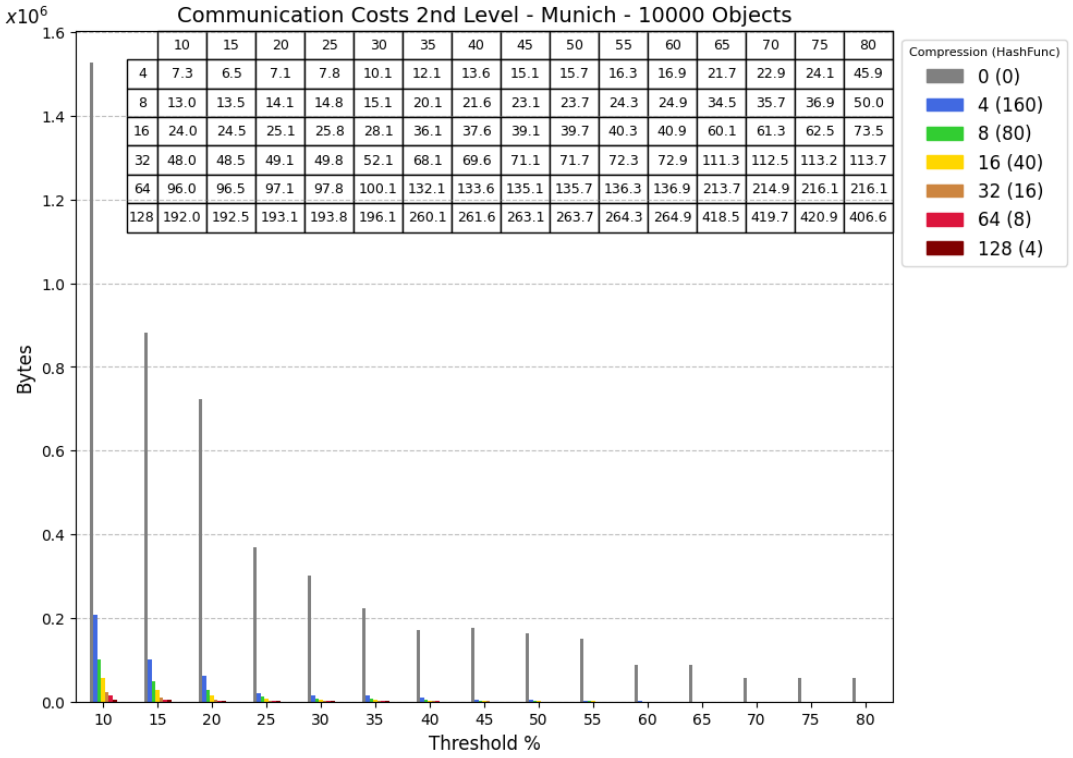
	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80
4	5.9	7.5	7.9	9.4	11.2	11.2	12.2	16.2	16.5	16.8	17.1	28.1	28.8	29.5	29.5
8	14.9	14.7	15.1	15.6	16.4	16.4	17.4	18.4	18.7	37.0	39.3	51.3	52.0	52.7	53.1
16	26.8	27.1	21.5	22.0	22.8	30.8	31.8	32.8	43.1	53.4	53.7	97.4	97.7	96.1	96.7
32	90.6	91.9	92.3	92.8	93.6	94.6	95.6	101.6	102.9	103.2	103.5	130.5	141.2	151.9	152.1
64	150.2	153.5	153.9	154.4	156.0	157.2	156.2	177.2	177.7	178.2	178.7	192.7	193.5	194.3	193.5

Figure 6.39: Communication costs 2nd Level - Munich network, 180 sensors, 2.5k tagIDs.



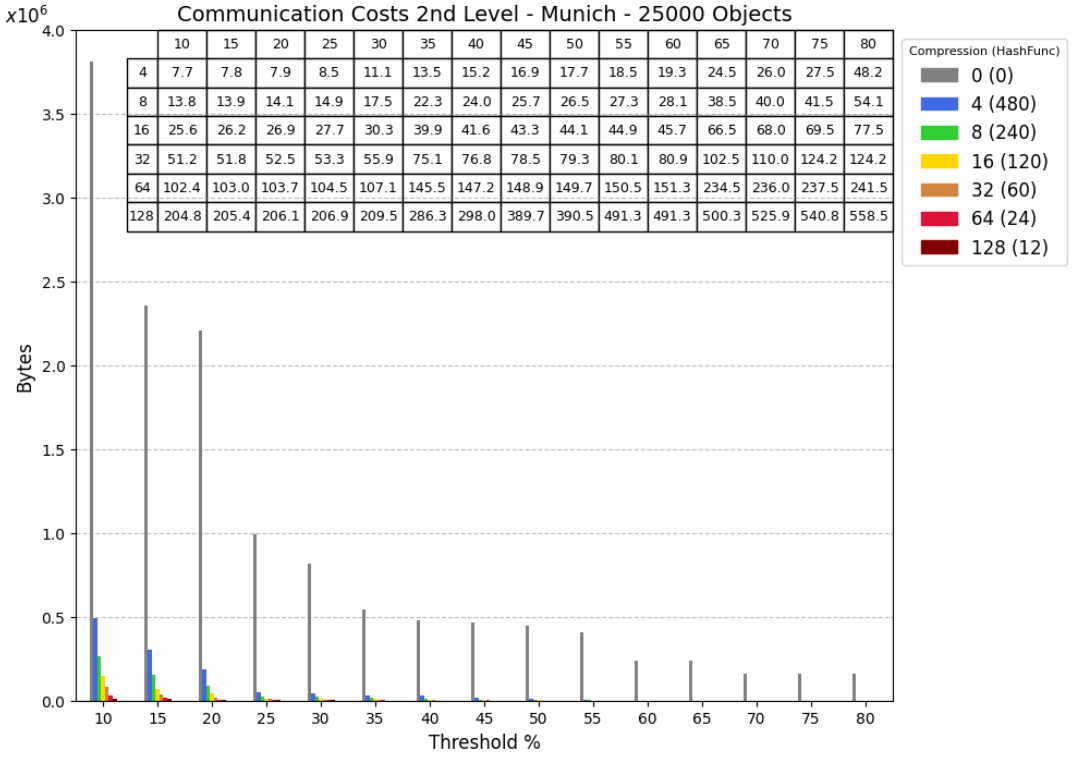
	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80
4	4.8	6.0	6.5	7.1	9.1	10.7	11.0	11.3	15.8	16.3	16.8	29.2	30.2	32.1	32.1
8	14.9	11.6	12.1	12.7	14.7	17.9	19.2	20.5	21.0	21.5	22.0	30.8	31.8	32.8	45.5
16	26.4	26.8	27.3	27.9	27.9	32.3	33.6	34.9	35.4	35.9	36.4	54.0	55.0	56.0	69.1
32	44.8	45.2	45.7	46.3	48.3	61.1	62.4	63.7	64.2	64.7	65.2	100.4	101.4	102.4	93.2
64	89.6	90.0	90.5	91.1	93.1	118.7	120.0	121.3	121.8	122.3	122.8	193.2	194.2	195.2	195.2
128	179.2	179.6	180.1	180.7	182.7	233.9	235.2	236.5	237.0	237.5	238.0	378.8	379.8	380.8	399.8

Figure 6.40: Communication costs 2nd Level - Munich network, 180 sensors, 5k tagIDs.



	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80
4	7.3	6.5	7.1	7.8	10.1	12.1	13.6	15.1	15.7	16.3	16.9	21.7	22.9	24.1	45.9
8	13.0	13.5	14.1	14.8	15.1	20.1	21.6	23.1	23.7	24.3	24.9	34.5	35.7	36.9	50.0
16	24.0	24.5	25.1	25.8	28.1	36.1	37.6	39.1	39.7	40.3	40.9	60.1	61.3	62.5	73.5
32	48.0	48.5	49.1	49.8	52.1	68.1	69.6	71.1	71.7	72.3	72.9	111.3	112.5	113.2	113.7
64	96.0	96.5	97.1	97.8	100.1	132.1	133.6	135.1	135.7	136.3	136.9	213.7	214.9	216.1	216.1
128	192.0	192.5	193.1	193.8	196.1	260.1	261.6	263.1	263.7	264.3	264.9	418.5	419.7	420.9	406.6

Figure 6.41: Communication costs 2nd Level - Munich network, 180 sensors, 10k tagIDs.



	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80
4	7.7	7.8	7.9	8.5	11.1	13.5	15.2	16.9	17.7	18.5	19.3	24.5	26.0	27.5	48.2
8	13.8	13.9	14.1	14.9	17.5	22.3	24.0	25.7	26.5	27.3	28.1	38.5	40.0	41.5	54.1
16	25.6	26.2	26.9	27.7	30.3	39.9	41.6	43.3	44.1	44.9	45.7	66.5	68.0	69.5	77.5
32	51.2	51.8	52.5	53.3	55.9	75.1	76.8	78.5	79.3	80.1	80.9	102.5	110.0	124.2	124.2
64	102.4	103.0	103.7	104.5	107.1	145.5	147.2	148.9	149.7	150.5	151.3	234.5	236.0	237.5	241.5
128	204.8	205.4	206.1	206.9	209.5	286.3	298.0	389.7	390.5	491.3	491.3	500.3	525.9	540.8	558.5

Figure 6.42: Communication costs 2nd Level - Munich network, 180 sensors, 25k tagIDs).

6.4 Second-Level Communication Costs and Multi-Factor Interactions

This section summarizes communication between Regional Leaders ($RegL_j$) and the Top Leader ($TopLeader$) in verifying cross-region Hot Motion Paths. We compare **Raw** and **MinHashed** data retrieval modes and examine how threshold settings, traffic volume, overlaps, and false positives collectively affect overhead.

6.4.1 Overview of Second-Level Queries

The Top Leader queries Region A and Region B if a hot path p_A in A might extend into p_B in B:

- **Raw Mode:** Requests full sets of distinct object IDs for each path/edge; overhead increases with set size.
- **MinHashed Mode:** Transmits compact MinHash signatures per path/edge to estimate intersection/union sizes.

MinHash avoids sending full sets, offering substantial size reduction—especially when object sets are large or repeated merges occur.

6.4.2 Factors Influencing Second-Level Overhead

Several interacting factors determine total communication cost:

1. **Threshold (t)**:
   - *Lower values:* More paths qualify as hot, increasing boundary checks and potential false positives in MinHashed mode.
   - *Higher values:* Fewer paths are considered, reducing checks and false positives; raw mode cost per path may rise due to higher local traffic volume.
2. **Traffic Volume (N) and Overlap (α)**: High object overlap across regions increases raw transmission size. Hashed mode remains constant due to fixed-size signatures.
3. **Compression Factor ($\mathcal{C}$)**: Hashed communication cost is reduced by a factor of $\mathcal{C}$, though repeated merges or approximate checks may slightly affect this ratio.
4. **Caching Potential**: Raw sets are often too large to cache at the Top Leader. MinHashed data, being small, can be reused efficiently across merges, reducing repeated transmission cost.

6.4.3 Suggestive Cost Expression

A general formulation can describe second-level communication cost:

$$\text{CommCost}_{\text{2-level}}(t,\ N,\ R,\ \alpha,\ \mathcal{C},\ \dots) = \Gamma(t, N, R, \alpha, \mathcal{C}, \text{etc.}), \tag{6.1}$$

with:

- t: hotness threshold,
- N: number of objects,
- R: number of regions,
- α: overlap factor,
- $\mathcal{C}$: compression factor,
- $\dots$: additional variables such as false positive rate or number of merges.

Although no closed form exists, Eq. (6.1) captures key dependencies:

1. Number of triggered merges (via),
2. Raw set size (N, α),
3. Hashed cost $\approx \dfrac{\text{raw size}}{\mathcal{C}}$,
4. False positives and caching feasibility.

Data-driven models may use this form to estimate or optimize the hashed/raw cost ratio in real systems.

6.4.4 MinHash and Elimination of Full Set Retrieval

MinHash eliminates the need to retrieve complete object sets. In raw mode, a path with thousands of object IDs requires full transfer. MinHash signatures, by contrast, are fixed-size and independent of cardinality. Thus, especially in large N or high-overlap scenarios, MinHash substantially reduces second-level communication cost.

Overall, second-level communication overhead is shaped by threshold tuning, object distribution, compression settings, and caching feasibility. MinHash provides a scalable alternative to raw transmissions, enabling efficient cross-region validation in sensor networks.

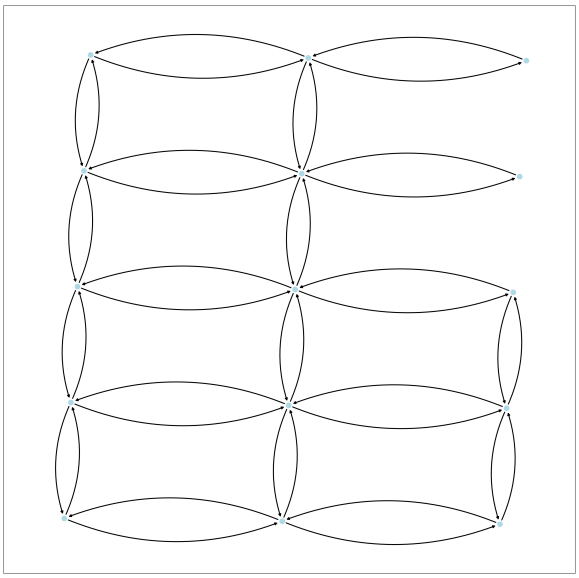
6.5 Experimental Results - San Francisco Network

To evaluate scalability, we conducted repeated experiments on road networks with diverse structural and geometrical configurations to exclude bias and to observe results independent of the characteristics of any specific network. We focus on a small area of San Francisco where RFID transponders are already deployed [13] .

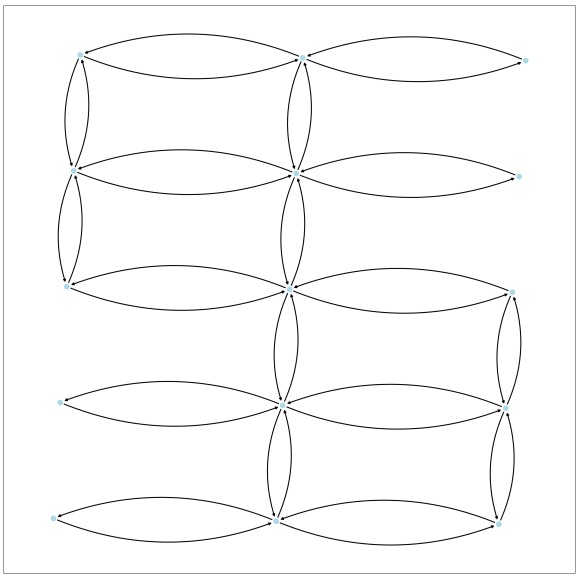
Figure 6.43: Area in San Francisco with 180 Edges

The analysis confirms that results from the first experiment in Munich are also observed in a distinct network, such as the one in San Francisco. Notably, each edge in the graph is bidirectional, with direction defined by the nodes—e.g., edge AB represents movement from node A to node B, while edge BA denotes the opposite direction.

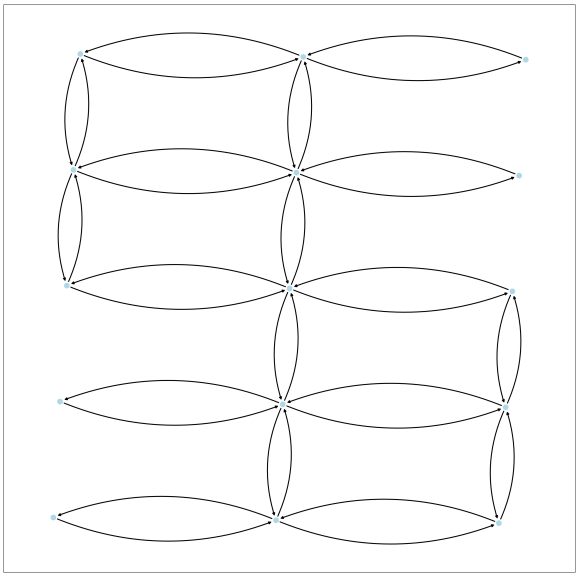
As shown in the figures below, we follow the same approach by dividing the network into four regions, each managed by a designated regional leader. Each region leader is responsible for overseeing 30 to 50 edges, ensuring a structured and efficient network partitioning.



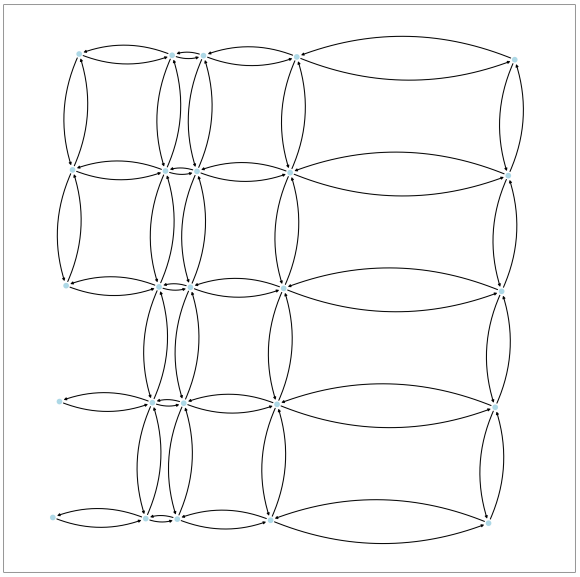
(a) Region 1



(b) Region 2



(c) Region 3



(d) Region 4

Figure 6.44: 4-Partitioning San Francisco: 4 Regions ($R_{g1}, R_{g2}, R_{g3}, R_{g4}$).

6.5.1 HoMoED Accuracy and 1st level Communication Costs

This highlights the robustness of our approach across varying network conditions. Specifically, higher number of hash functions provide stable and high accuracy. However, under very high traffic loads, the communication costs—particularly at the first level—may become a limiting factor. The root cause of this phenomenon can be attributed to collisions (NACK bytes) occurring during transmission in our RFID network, as outlined in 6.1.5.

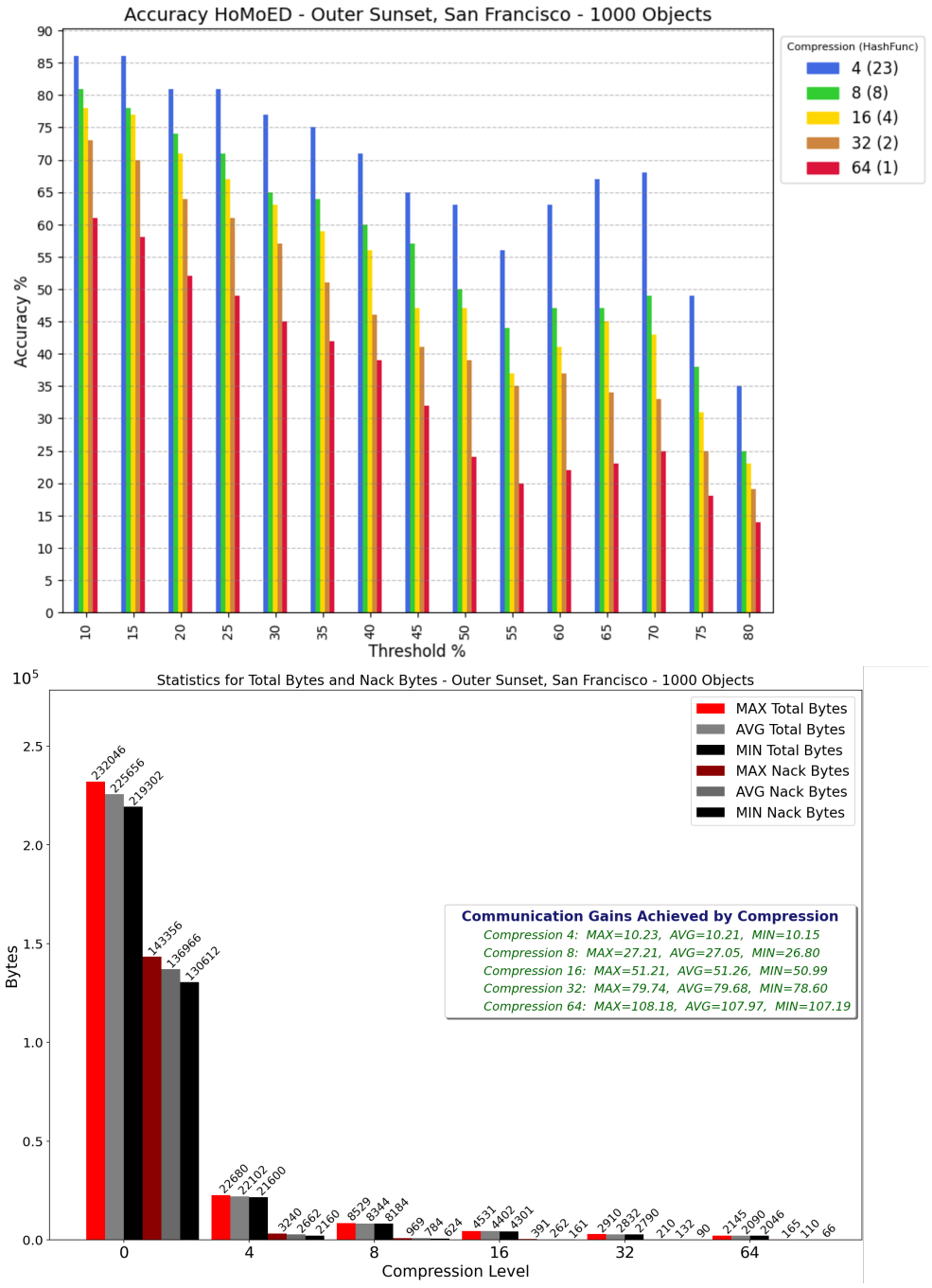


Figure 6.45: HoMoED Accuracy and 1st Level Communication Costs,San Francisco, 1k tagIDs.

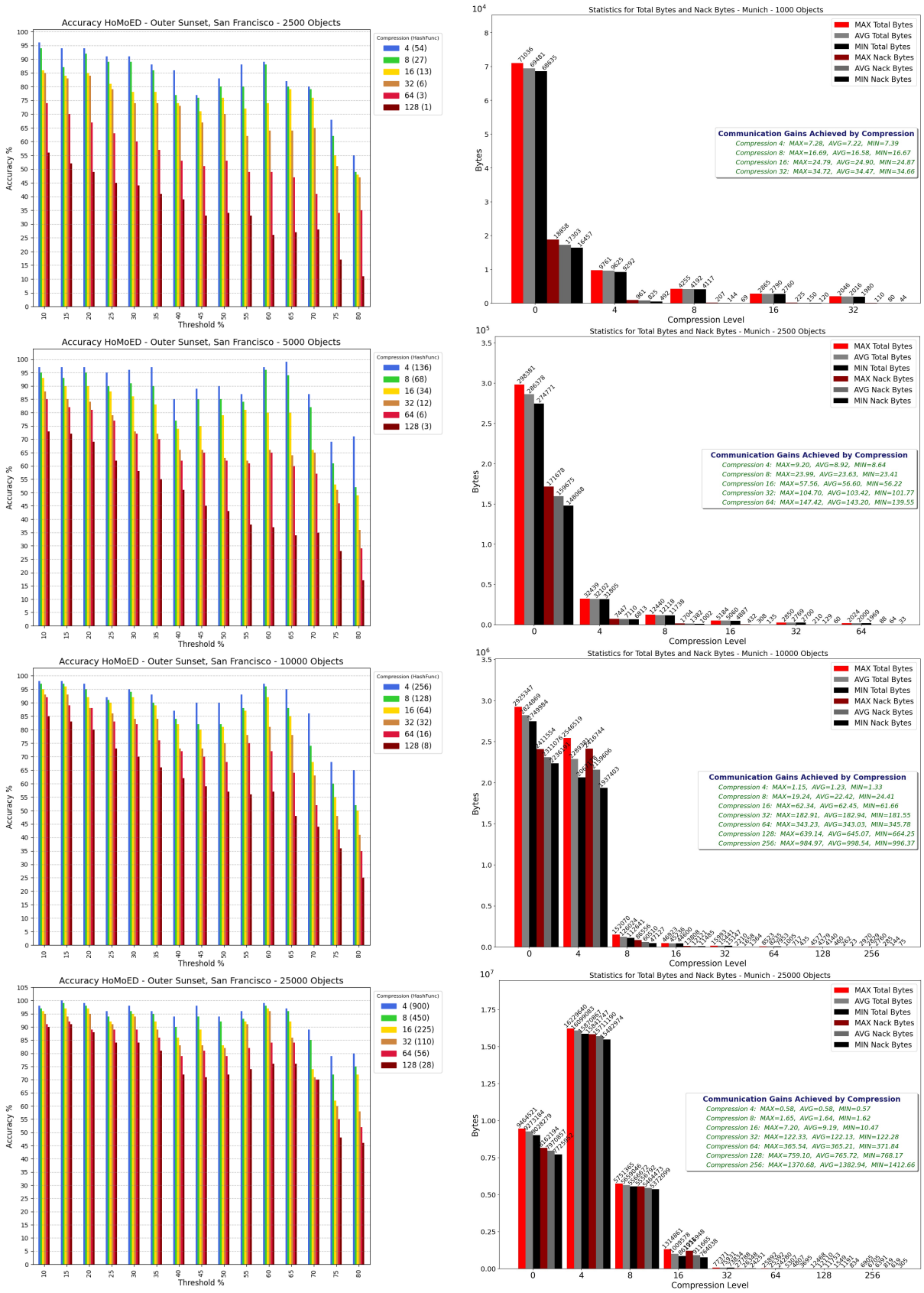


Figure 6.46: HoMoED Accuracy and 1st Level Communication Costs, San Francisco

6.5.2 HoMoPaD Results - Real Scenario

Additionally, the HoMoPaD algorithm exhibits similar trends to the Munich network, highlighting the trade-off between accuracy, thresholds, and number of hash functions / compression rates across different network conditions.

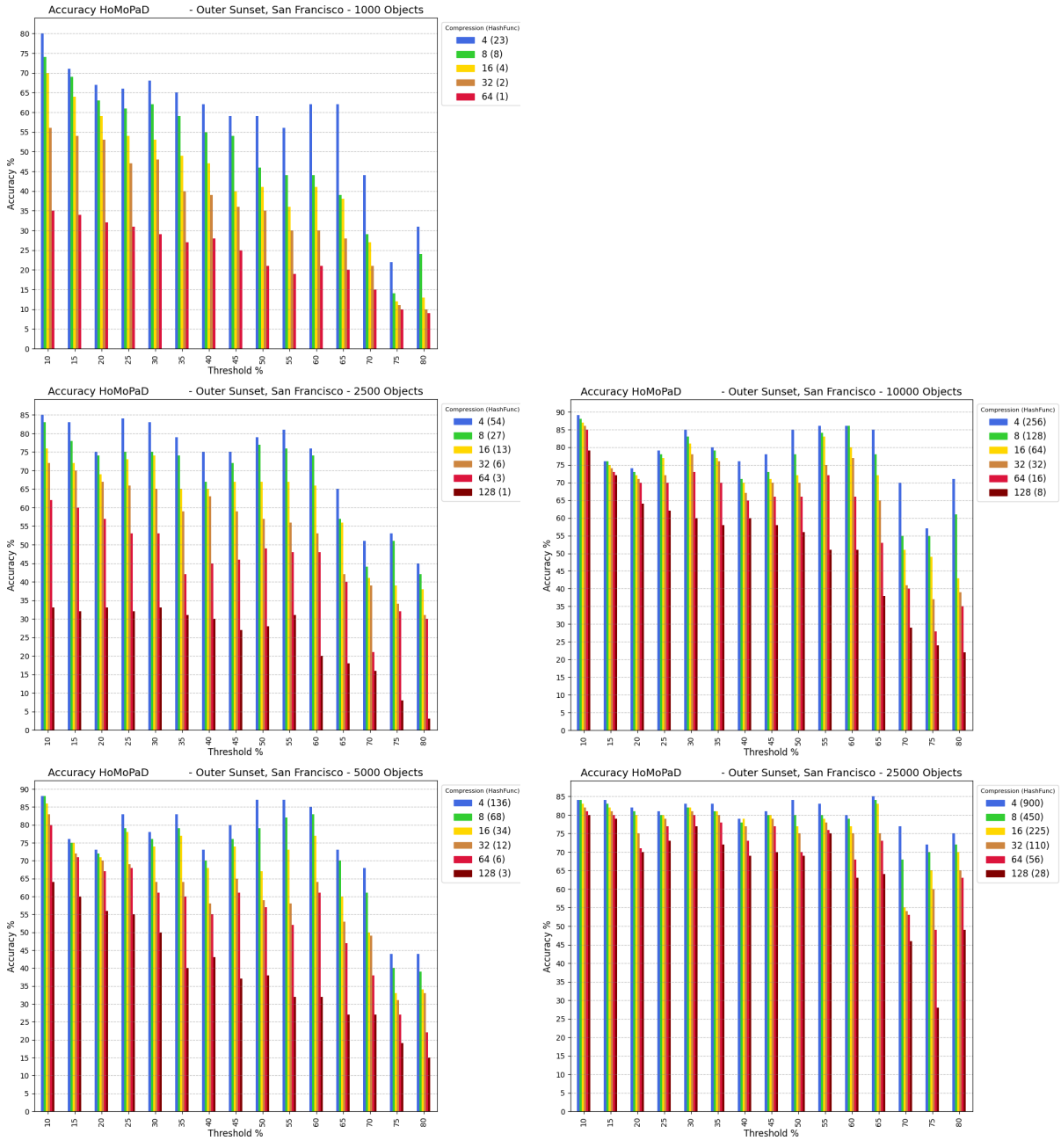


Figure 6.47: HoMoPaD Accuracy , San Francisco

6.5.3 HoMoPaD Communication costs 2nd Level - Real Scenario

As previously mentioned, second-level communication costs cannot be universally accounted for, as they depend on the specifics of each experiment. However, what remains observable is the relative compression of communication costs: as the number of hash functions increases, the compression decreases accordingly. The following plot presents the second-level communication costs observed in the San Francisco network.

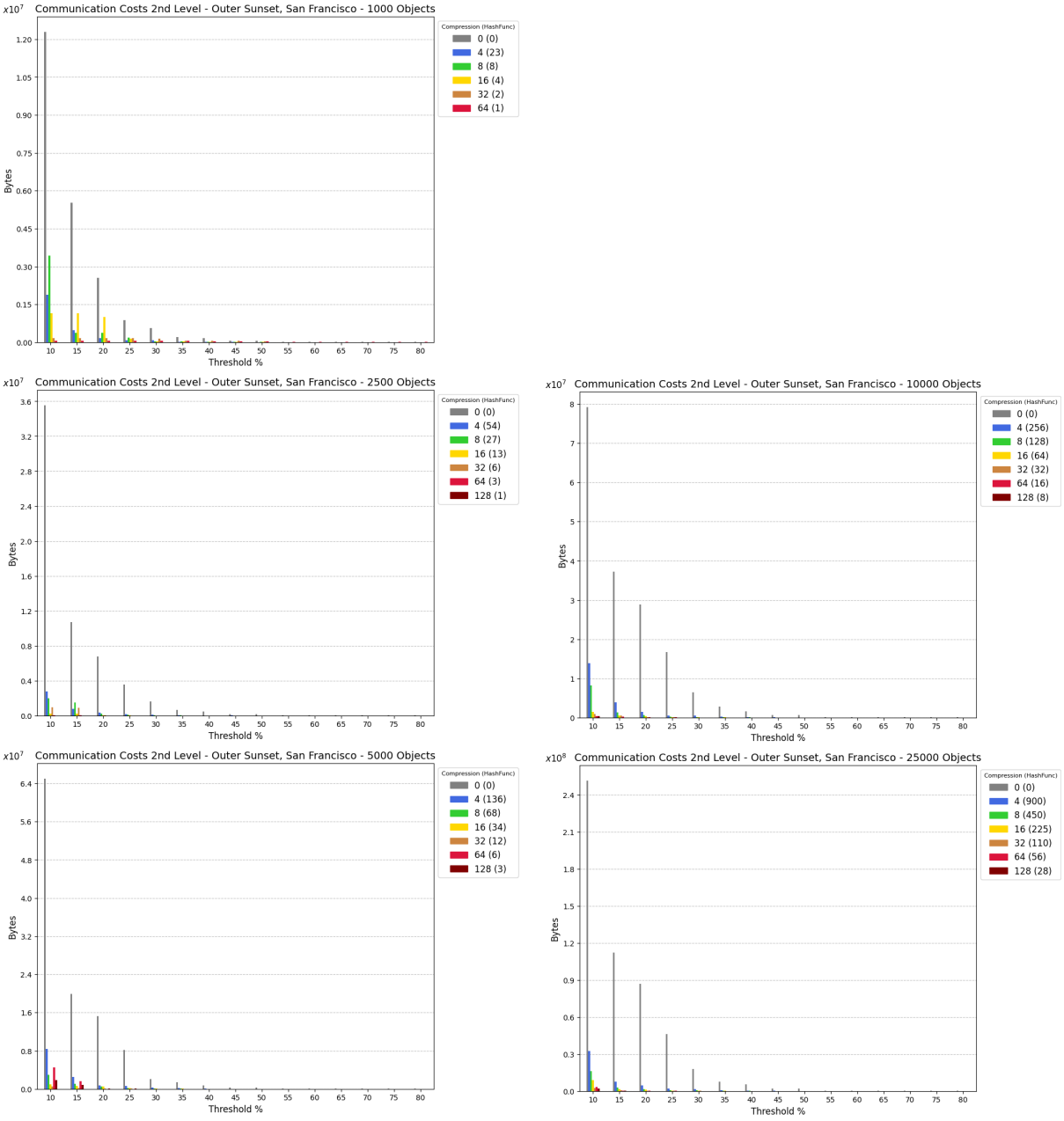


Figure 6.48: HoMoPaD 2nd Level Communication Sosts , San Francisco

6.6 Experimental Results - Chania Network

Chania, a historic city in Greece, boasts a richly intricate old town with a labyrinth of narrow streets and centuries-old infrastructure. This unique network presents significant challenges, highlighting the complexities of managing urban environments in heritage-rich areas. The city's layout, shaped by organic growth over centuries, is not well-suited for modern vehicular traffic. As a result, Chania stands out as an ideal candidate for future real-world applications, moving beyond theoretical models to address these practical urban challenges.

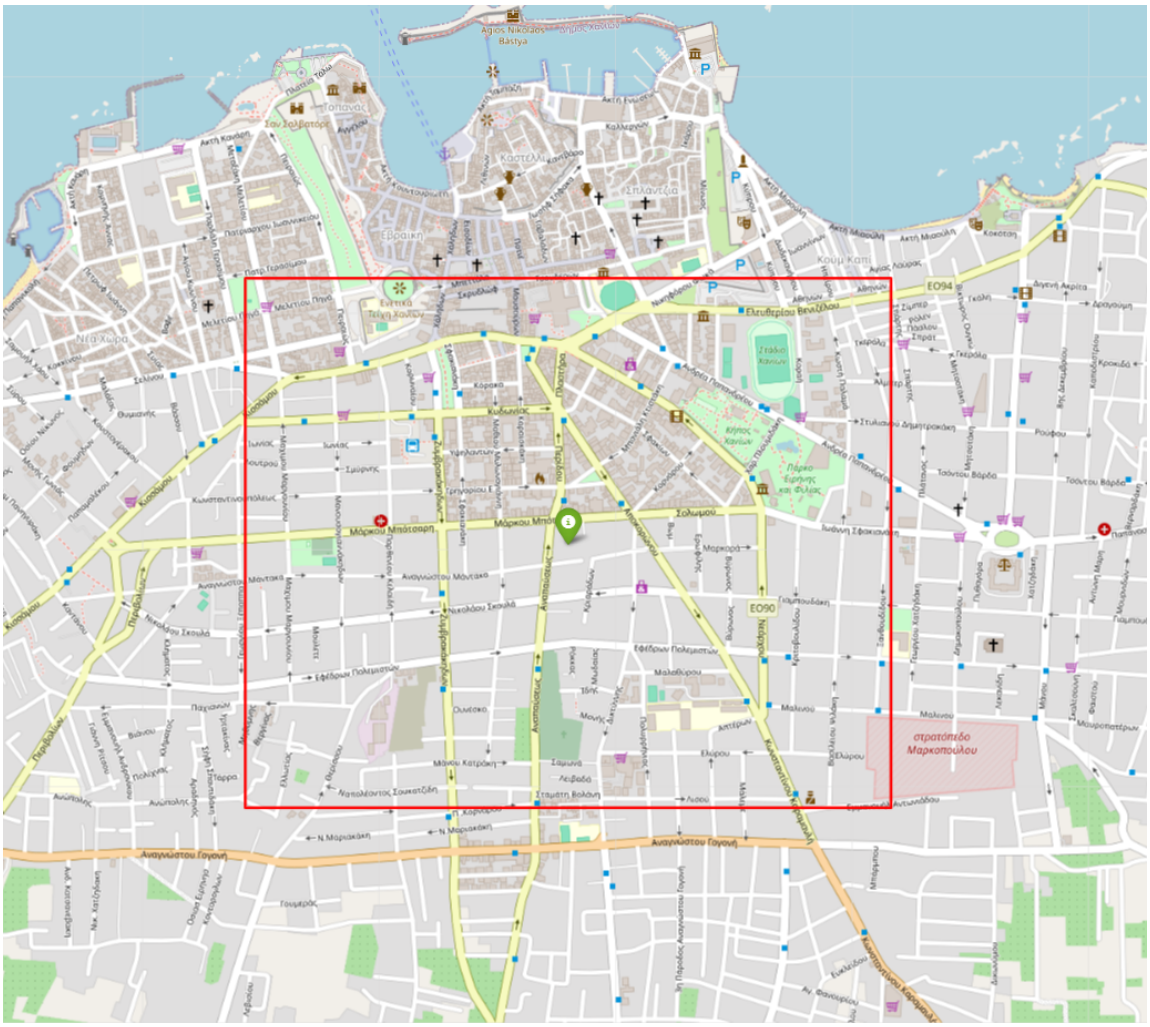


Figure 6.49: Area in Chania with 180 Edges

During the summer months, Chania experiences a significant surge in traffic due to an influx of tourists, further straining the existing road network. The combination of local commuters, service vehicles, and visitor traffic often leads to congestion hotspots, complicating navigation and logistics. This seasonal variation introduces an additional layer of complexity, requiring adaptive traffic management strategies to maintain efficient flow and minimize delays.

6.6.1 HoMoED Accuracy and 1st level Communication Costs

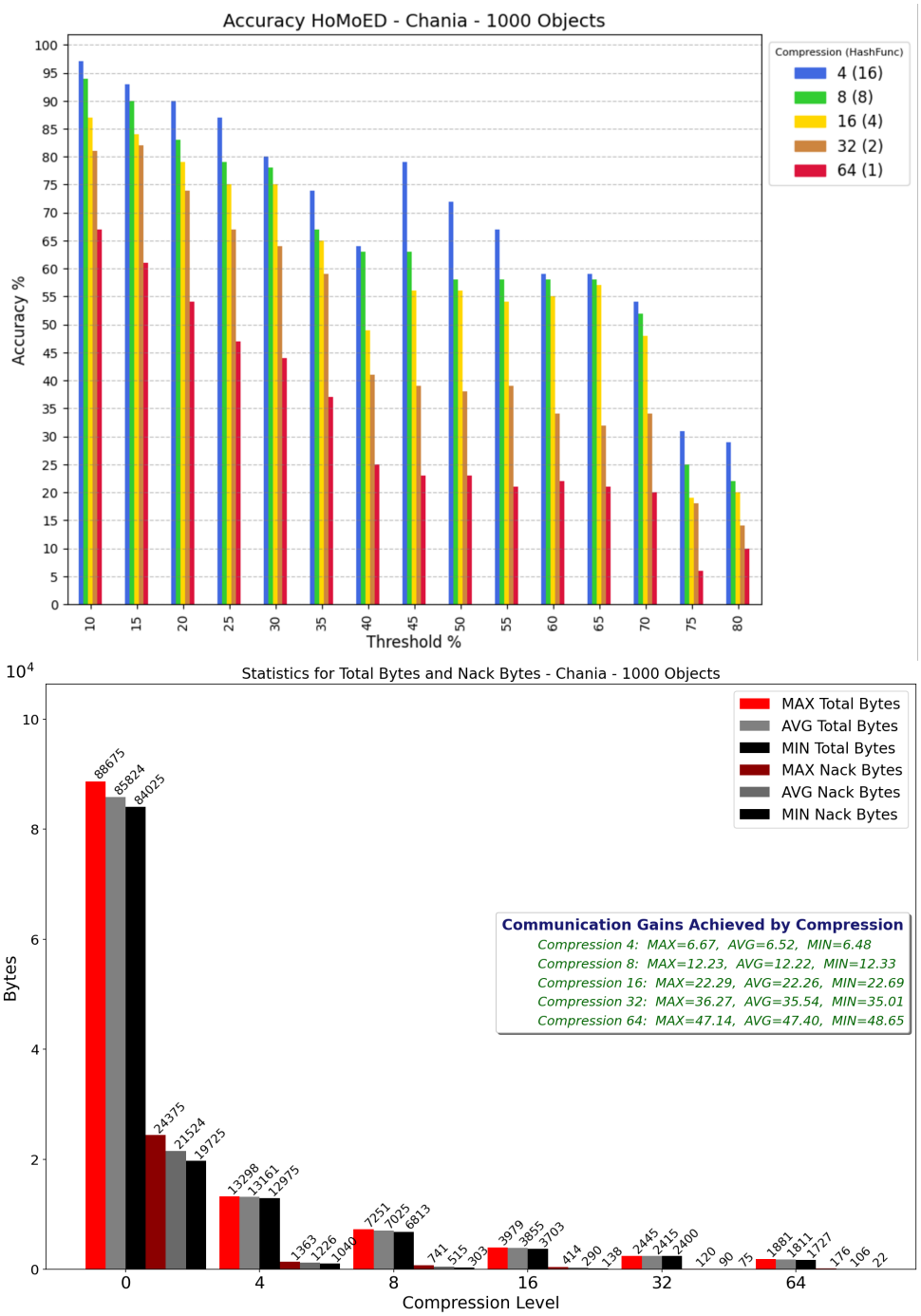
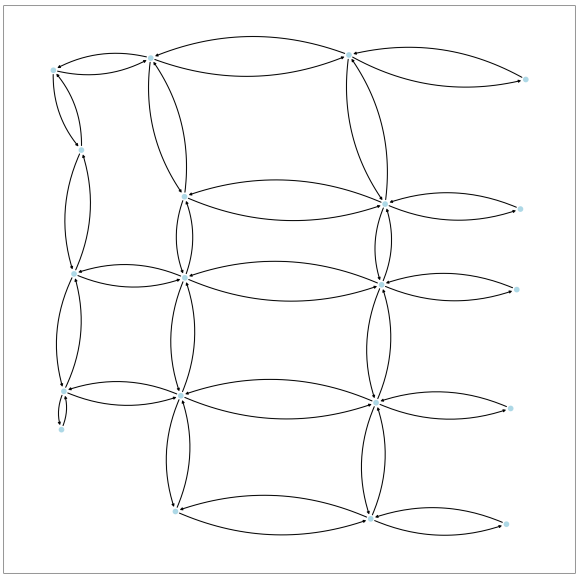
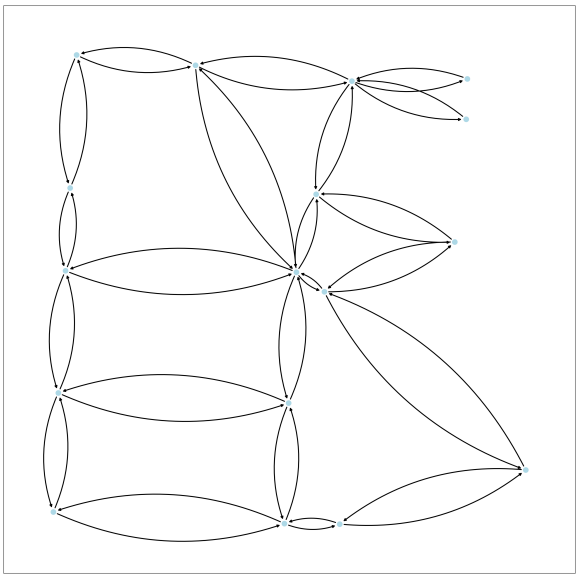


Figure 6.50: HoMoED Accuracy and 1st Level Communication Costs , Chania , 1k tagIDs.

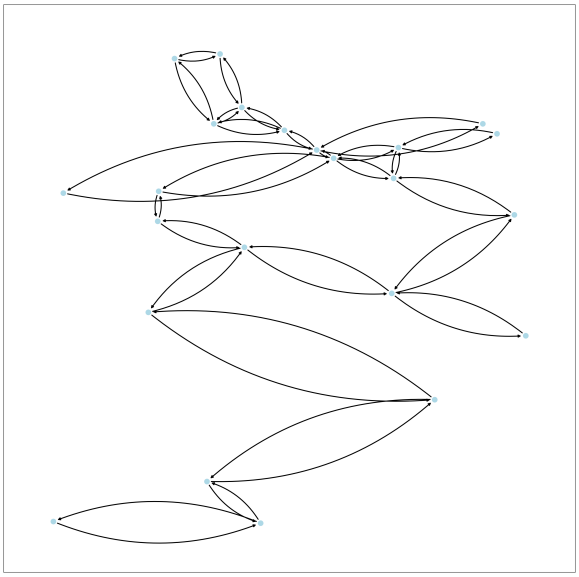
Accuracy and communication costs at the first level follow a similar trend, confirming that these findings hold regardless of network structure and connectivity. As shown in Figure 6.50, the communication gain ratios remain consistent, while accuracy drops notably with each threshold increase, reflecting the system's sensitivity to compression (number of hash functions). Higher hash function counts at the same threshold improve accuracy by reducing the compression ratio and icnreasing communications costs.



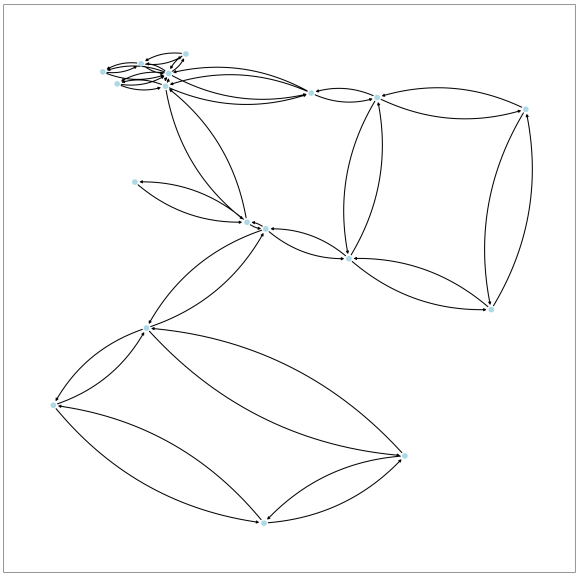
(a) Region 1



(b) Region 2



(c) Region 3



(d) Region 4

Figure 6.51: 4-Partitioning Chania: 4 Regions ($R_{g1}, R_{g2}, R_{g3}, R_{g4}$).

We observe similar trends based on the different traffic volumes. For instance, compression rates as low as 4:1 maintaining stable and high accuracy under moderate traffic volumes(2,500 or 5,000 moving objects). However, at very high traffic loads, communication costs at the first level can become a limiting factor, due to increased collisions (NACK bytes) as has been explained. Compression rates of 16:1 and 32:1 consistently maintain high accuracy for thresholds up to 60–70%, while higher ratios (64:1 and 128:1) continue to sustain strong accuracy as traffic volume increases or under lower thresholds and reduced traffic loads, all while achieving significant communication efficiency.

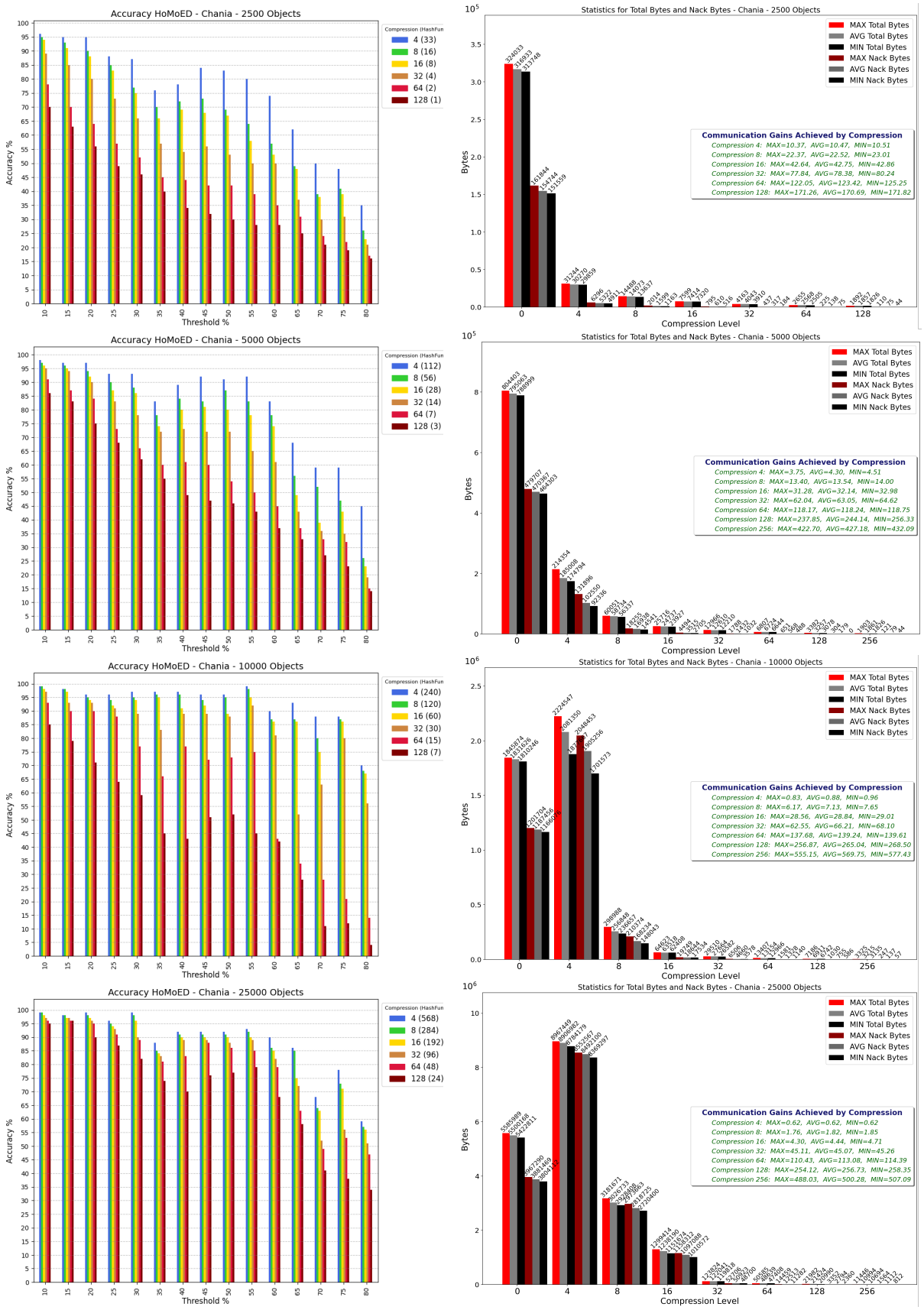


Figure 6.52: HoMoED Accuracy and 1st Level Communication Costs , Chania

Considering the implications of time-based sensors (3.3.1), some sensors record fewer objects than others. After the sensors with fewer recorded objects transmit successfully, they free up bandwidth for those with more recorded data. However, when Minhashing is applied (5.2), all sensors generate a uniform-length MinHash signature, complicating transmission and increasing collisions. For instance, in the Chania Network, compression 4:1 already led to higher communication costs than raw data transmission for a traffic volume of 10,000 objects. This highlights the uncertainty of the exact traffic volume at which these turning points occur, though they undoubtedly exist as traffic increases.

6.6.2 HoMoPaD Results - Real Scenario

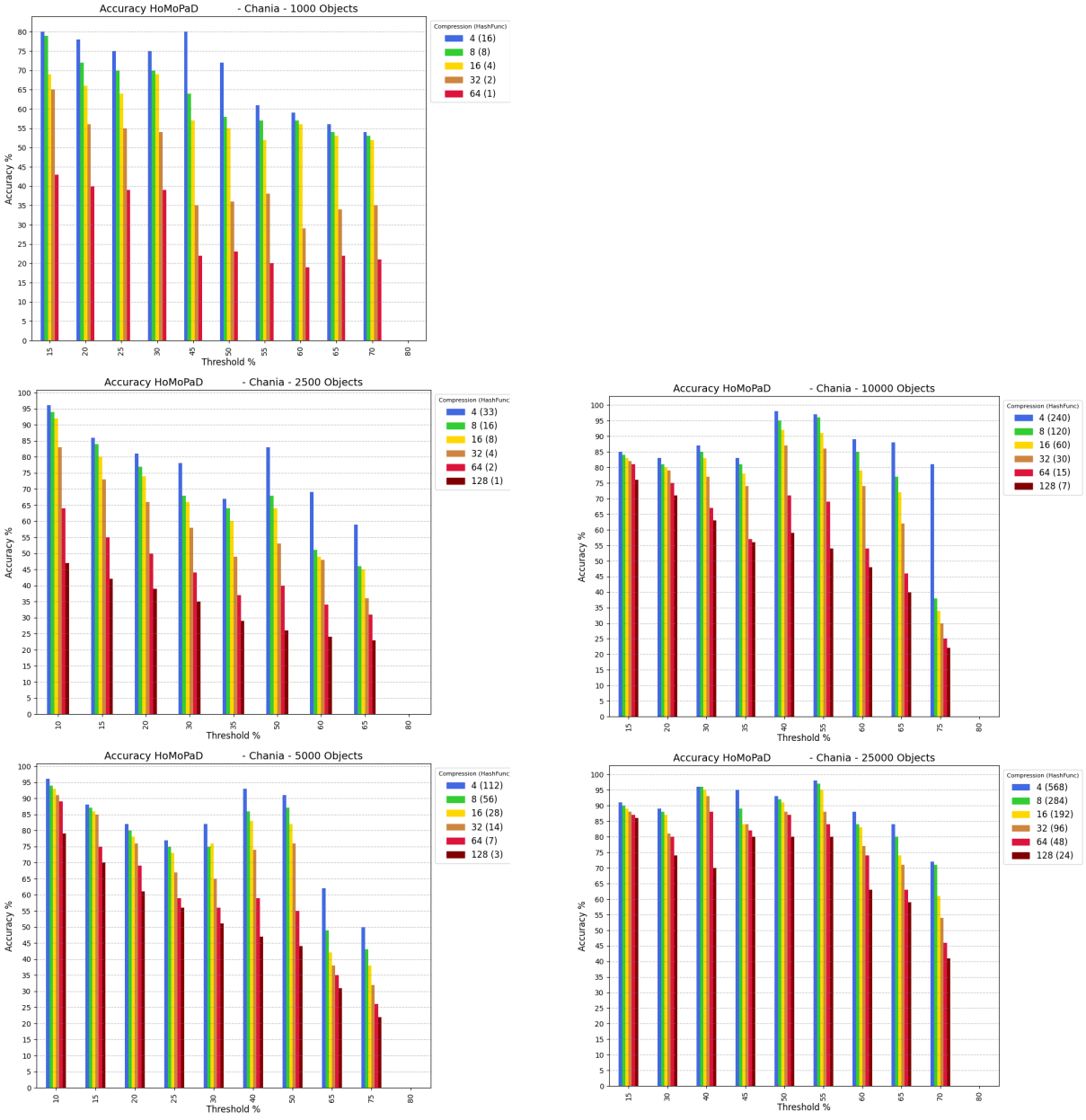


Figure 6.53: HoMoPaD Accuracy , Chania

6.6.3 HoMoPaD Communication costs 2nd Level - Real Scenario

As discussed earlier, second-level communication costs are experiment-dependent and cannot be generalized. However, a clear trend remains: increasing the number of hash functions leads to lower compression, directly affecting communication costs. The next plot illustrates the second-level communication costs recorded in the Chania network.

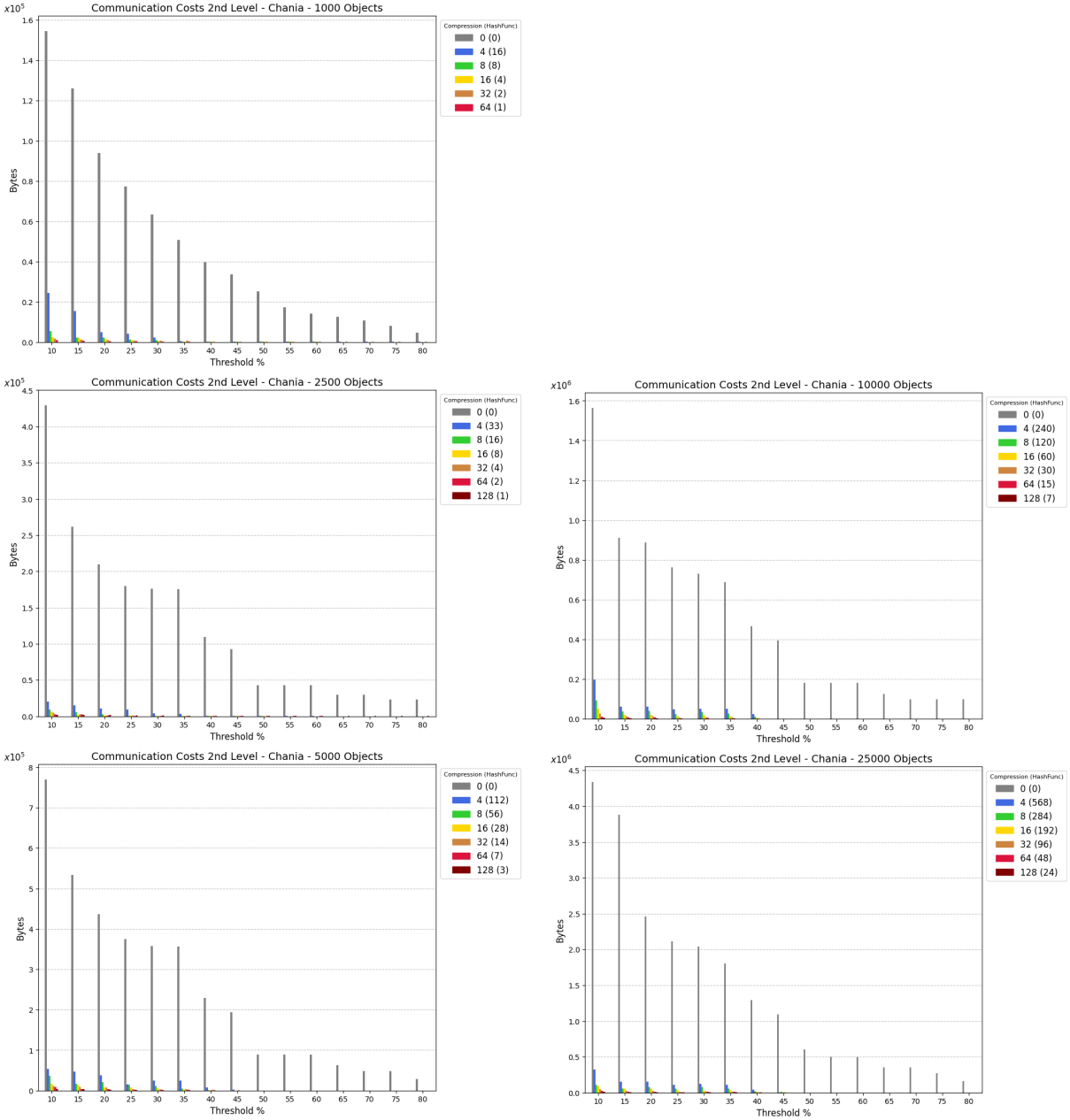


Figure 6.54: HoMoPaD 2nd Level Communication Sosts , Chania

6.6.4 Conclusion

We conclude our experimental and theoretical analysis by summarizing key insights from HoMoED and HoMoPaD. Our findings reveal relationships between number of hash functions, compression, accuracy, threshold selection, and communication costs, validating the effectiveness of MinHash-based approximations in distributed road networks.

- **Accuracy in HoMoED and HoMoPaD depends on k, not compression:** Hoeffding's bound confirms that accuracy is solely governed by the number of hash functions (k), independent of raw set length or compression rate. Since k also determines communication costs, it remains the key factor in balancing accuracy and resource efficiency.

- **MinHash-based Jaccard similarity approximates HoMoPaD well at higher thresholds.** At low thresholds, the denominator includes almost all edges, increasing noise. As the threshold rises, it shifts toward high-traffic edges, reducing inaccuracies. The first-level communication protocol naturally limits universe expansion, further reinforcing MinHash approximation reliability.

- **New Error Bound for Jaccard Similarity Estimation:** Using Hoeffding's inequality, we establish a more precise bound on the accuracy of Jaccard similarity estimation via MinHash. This result ensures that the estimated Jaccard index remains within allowed error $\pm\epsilon$ of the true value with probability at least $1-\delta$, given by: $\mathrm{k} \geq \mathrm{O}\left(\frac{\ln(2/\delta)}{2\epsilon^2}\right)$). It confirms that accuracy depends purely on k, independent of raw set size or compression factor. Additionally, while optimizing communication costs, the condition $k < |O_{e_i}|$ must hold to ensure compression benefits without exceeding transmitted set sizes.

- **Application of Cohen's Estimation in HoMoED:** The CCM approach preserves local traffic accuracy while leveraging Cohen's estimation for scalable global approximations. Using a direct counter for the numerator ensures precise sensor-reported traffic, preventing discrepancies between local (A-LESE) and global (G-LESE) estimations. The counter adds minimal overhead—only 1-2 bytes—making it a negligible cost for improved HoMoEdge detection accuracy.

- **Alternative (Hoeffding-based) probabilistic bound for Cohen's estimator:** By integrating Hoeffding's inequality with Taylor's first-order expansion, we establish a new route for the probabilistic bound for Cohen's size estimator. This ensures that the relative error remains within $\pm\epsilon$ with probability at least $1-\delta$, leading to $\mathrm{k} = \mathrm{O}\left(\frac{\ln(2/\delta)}{2\epsilon^2}\right)$. This result improves the Central Limit Theorem's (CLT) error bound, requiring fewer hash functions while preserving high accuracy. This bound achieves up to three times fewer hash functions than CLT for the same confidence level, confirming Cohen's Chebyshev-based error bounding.

- **Errors Propagation from HoMoED to HoMoPaD:** The investigation into HoMoPath detection highlights that errors from the HoMoEdge detection phase can severely impact subsequent detection processes. Maintaining a HoMoED accuracy of 60-70% is critical for ensuring acceptable performance in HoMoPaD.

- **HoMoPaD remains stable even with moderate HoMoED errors.** As threshold increases, HoMoPaD (real) approaches HoMoED performance, confirming that early-stage errors do not degrade the final results significantly. In some cases, HoMoPaD (real) even outperforms HoMoED, aligning with the transitive closure principle, where misclassified edges are still likely to form meaningful motion paths due to road constraints.

- **Threshold selection influences accuracy and communication efficiency.** Higher thresholds filter out weaker edges, improving HoMoPaD accuracy but reducing HoMoED's ability to detect low-traffic edges. Lower thresholds increase coverage but introduce noise, emphasizing the need for fine-tuning the *minHoMoSup* paramete

- **Distributed constraints ensure practical system scalability and improve MinHash-based approximation.** The inherent limitations of first-level communication prevent excessive sensor collisions and maintain a manageable universe size. Additionally, the distributed nature of the system aids HoMoPaD's alignment with MinHash-based Jaccard similarity by dividing the road network into smaller parts, each managed by a regional leader. This partitioning reduces the overall noise in the denominator and ensures that Jaccard estimations remain more precise within localized traffic patterns.

- **Compression affects communication gain in a non-linear manner.** A *compression turning point* exists where communication gains surpass compression ratios, after which further gains stabilize. Below this point, excessive compression increases transmission costs due to overhead, while beyond it, reducing hash functions optimally balances accuracy and efficiency.

- **Effects of Randomization in Hash Functions:** The results reveal that variability in hash function selection introduces inconsistencies in detection accuracy. The random nature of hash functions underlines the potential benefit of employing deterministic or adaptive hashing techniques to enhance reliability and performance.

- **Code:** The complete implementation supporting all experiments and results presented in this work is publicly available at:
`https://github.com/christodouloskampanis/HoMoPaD`

Chapter 7

Future Research

Future research should address several key areas to further enhance the performance and applicability of the methodologies presented.

Randomized Selection of Hash Functions

A significant area of interest lies in improving the randomized selection of hash functions to optimize accuracy, even when limited to a single random seed or family of hash functions. Exploring techniques such as adaptive hashing, dynamic seed generation, or hybrid hashing approaches may reduce variance in accuracy results. Additionally, leveraging machine learning models to select the most suitable hash functions based on prior network conditions could further enhance accuracy without increasing computational overhead.

Communication Cost Optimization

Reducing communication costs remains a critical objective, particularly at the first and second levels of the network hierarchy. One approach involves optimizing data aggregation techniques at the sensor level, minimizing redundant transmissions. Implementing more efficient encoding or compression strategies could also help reduce the volume of transmitted data, thereby lowering communication overhead. Additionally, exploring probabilistic data structures that minimize the need for repeated acknowledgments (ACK/NACK) may further reduce costs.

Alternative Communication Models for Second Level (RegLeader to TopLeader)

Investigating alternative communication models between the Regional Leaders (RegLeader) and Top Leader offers another promising avenue. Techniques such as multi-hop communication, direct relay mechanisms, or clustering-based approaches could streamline data transfer, reducing latency and overhead. Exploring decentralized methods, where RegLeaders share partial aggregate data directly with peers before reaching the Top Leader, could also balance communication load and enhance scalability.

FCM Approach and Cardinality Estimation

Another potential direction involves adopting the Full Stack Cardinality Minhash (FCM) approach, which eliminates the need to transmit extra counters reflecting the number of real objects recorded by each sensor. Instead, the focus would shift to estimating Atomic Cardinality (A-LESE) for individual sets and Global Cardinality (G-LESE) for the entire system. This estimation-based method enhances data transmission efficiency and scalability, as outlined in Section 5.7 and detailed in Tables 5.6–5.9, with further insights provided in the accompanying example. Future research could investigate the trade-offs between precision and overhead in adopting the FCM approach, while also exploring its interplay with the JISE method as an alternative for estimating the cardinality of the entire union.

JISE as a Heuristic Tool

A particularly interesting area for future exploration is the use of JISE (Joint Intersection Set Estimator) as a heuristic approach to identify overlaps between different data sets. Our experiments indicate that the overlap and the number of hash functions used significantly affect JISE's performance, offering valuable insights into its behavior.

For instance, consider astrophysical data collected from multiple observatories regarding black hole properties. Such overlap could reveal whether the sources provided identical or highly similar results, reinforcing the validity of critical scientific observations.

Also n the medical and genetic fields, for example, DNA samples from patients diagnosed with a specific disease could be analyzed to identify common patterns, uncovering shared traits associated with the disease, potentially advancing diagnostic and therapeutic approaches.

JISE could be employed to detect consistent behaviors in repeated experiments, such as large-scale simulations in physics or computational biology, where detecting invariants across massive datasets is crucial. Imagine conducting a simulation or scientific test millions or even billions of times. The ability to rapidly identify overlaps between repeated trials would allow for quick detection of constant patterns, ensuring the reliability of the experiment. T

In summary, JISE's capacity to identify overlaps positions it as a versatile tool for enhancing scientific validation, pattern recognition, and experimental consistency.

Adaptive Sensor Activation and Compression Optimization

Future research could explore leveraging machine learning or historical data to optimize sensor activation times, reducing energy consumption in battery-powered sensors. For instance, during low-traffic periods, such as nighttime, sensors could operate at lower frequencies, preserving battery life. Additionally, predictive models could estimate network-wide traffic volume, enabling dynamic compression adjustments based on the number of hash functions. This would prevent scenarios where attempted compression inadvertently increases communication overhead, ensuring efficiency without exceeding raw data transmission costs.

Bibliography

[1] L. Zhang and Z. Wang, "Integration of RFID into Wireless Sensor Networks: Architectures, Opportunities and Challenging Problems" *Proceedings of GCCW*, pp. 463–469, 2006.

[2] M. Buettner, B. Greenstein, D. Wetherall, and J. Smith, "Revisiting Smart Dust with RFID Sensor Networks" *Proceedings of HotNets*, 2008.

[3] H. Liu, M. Bolic, A. Nayak, and I. Stojmenovic, "Integration of RFID and Wireless Sensor Network" in D. Agrawal (Ed.), *World Scientific Publishing Company*, 2008.

[4] X. Li, J. Han, J.-G. Lee, and H. Gonzalez, "Traffic Density-Based Discovery of Hot Routes in Road Networks" in *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 441–449, 2007.

[5] Q. Liu, S. Jiang, and T. Wang, "LiveMap: Real-Time Dynamic Map in Automotive Edge Computing" , 2020.

[6] S. Mukherjee, R. C. Shah, and K. Ramchandran, "SPIRE: Efficient Data Inference and Compression over RFID Streams" *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 1, pp. 141–155, 2012.

[7] Y. Zhang, H. Zhu, and X. Hu, "Intelligent Traffic Management System Based on WSN and RFID" in *Proceedings of IEEE International Conference on Green Computing and Communications*, pp. 773–776, 2010.

[8] Y. Zhuang, M. Zhang, and Z. Xu, "Detecting Traffic Hot Spots Using Vehicle Tracking Data" *SPIE Proceedings*, vol. 9901, 2017.

[9] T. Nie, X. Li, and Y. Wang, "Towards Better Traffic Volume Estimation: Jointly Addressing the Underdetermination and Nonequilibrium Problems with Correlation-Adaptive GNNs" , 2023.

[10] W. Ding, L. Chen, and F. Zhao, "Phased Deep Spatio-Temporal Learning for Highway Traffic Volume Prediction" , 2023.

[11] Y. Li, C. Wu, and D. Song, "A City-Wide Traffic Volume Estimation Method Based on Graph Autoencoder" *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[12] B. Sambana and G. Srisudha, "Vehicle Tracking and Traffic Detection System using Internet of Things" *Advanced Science and Technology Letters*, vol. 147, pp. 292–296, 2017.

[13] S. Subramaniam and D. Gunopulos, "A Survey of Stream Processing Problems and Techniques in Sensor Networks" in *Data Streams: Models and Algorithms*, 2007.

[14] D. Sacharidis, K. Patroumpas, M. Terrovitis, et al., "On-line discovery of hot motion paths" *Proceedings of EDBT*, 2008.

[15] Y. Zhang, "RFID-based tracking in supporting real-time urban traffic information" *Proceedings of NCM*, 2009.

[16] M. S. Chen, J. S. Park, and P. Yu, "Efficient Data Mining for Path Traversal Patterns" *IEEE Transactions on Knowledge and Data Engineering*, vol. 10, no. 2, pp. 209–221, 1998.

[17] Y. Xiao and M. H. Dunham, "Efficient mining of traversal patterns" *Data Knowledge Engineering*, vol. 39, no. 2, pp. 191–214, 2001.

[18] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation" *Proceedings of SIGMOD*, 2000.

[19] R. J. Bayardo, "Efficiently mining long patterns from databases" *Proceedings of SIGMOD*, 1998.

[20] E. Cohen, "Size-estimation framework with applications to transitive closure and reachability" *Journal of Computer and System Sciences*, vol. 55, no. 3, pp. 441–453, Dec. 1997.

[21] E. Cohen, M. Datar, S. Fujiwara, et al., "Finding Interesting Associations without Support Pruning" *Proceedings of ICDE*, 2000.

[22] E. Cohen, "MinHash Sketches: A Brief Survey" June 2016

[23] A. Azizi, "Introducing a Novel Hybrid Artificial Intelligence Algorithm to Optimize Network of Industrial Applications in Modern Manufacturing" *Complexity*, vol. 2017, Article ID 8728209, 18 pages, 2017.

[24] Z. Li, C. He, J. Li, X. Huang, et al., "RFID reader anti-collision algorithm using adaptive hierarchical artificial immune system" *Expert Systems with Applications*, vol. 41, no. 2, pp. 341–350, Feb. 2014.

[25] A. Z. Broder, "On the resemblance and containment of documents" in *Proceedings of the Compression and Complexity of Sequences*, IEEE, 1997, pp. 21–29.

[26] W. Hoeffding, "Probability inequalities for sums of bounded random variables" *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 13–30, 1963.

[27] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications" in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys '03)*, 2003.

[28] A. V. Aho, "The transitive reduction of a directed graph," *SIAM Journal on Computing*, vol. 1, no. 2, pp. 131–137, 1972.

[29] K. Beyer, J. Goldstein, and R. Ramakrishnan, "Set similarity search beyond MinHash," *arXiv preprint arXiv:1612.07710*, 2014.

[30] P. Li, K. Church, and T. Hastie, "Min-wise independent permutations," in *Proceedings of the 23rd International Conference on Machine Learning*, 2006.

[31] A. Z. Broder, M. S. Charikar, M. Mitzenmacher, and R. Panigrahy, “On the resemblance and containment of documents,” in *Proceedings of Compression and Complexity of Sequences 1997 (SEQUENCES '97)*, pp. 21–29, 1997.

[32] Beta Function properties and applications: Gauss, C. F. (1812). "Disquisitiones generales circa series infinitas." *Commentationes Societatis Regiae Scientiarum Gottingensis Recentiores*, Vol. 2.

[33] Johnson, N. L., Kotz, S., Balakrishnan, N. (1995). "Continuous Univariate Distributions." *Journal of the American Statistical Association*, 90(432), 1341-1349.

[34] Billingsley, P. (1995). "Probability and Measure." *John Wiley & Sons*, 3rd edition, New York. [ISBN: 978-0-471-00710-4]