Optimizing Operators for Temporal and Spatiotemporal Data

by

Rakan A. Alseghayer

B.Sc., King Saud University, 2008

M.Sc., University of Pittsburgh, 2013

Submitted to the Graduate Faculty of the

Dietrich School of Arts and Sciences in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2023

UNIVERSITY OF PITTSBURGH

DIETRICH SCHOOL OF ARTS AND SCIENCES

This dissertation was presented

by

Rakan A. Alseghayer

It was defended on

August 3, 2023

and approved by

Panos K. Chrysanthis, Department of Computer Science

Alexandros Labrinidis, Department of Computer Science

Kirk Pruhs, Department of Computer Science

Mohamed Sharaf, Department of CS & SE, United Arab Emirates University

Constantinos Costa, Department of CS, University of Cyprus & Rinnoco Ltd, Cyprus

Optimizing Operators for Temporal and Spatiotemporal Data

Rakan A. Alseghayer, PhD

University of Pittsburgh, 2023

Mobile devices and IoT technologies have become highly available, which led to the development of smart solutions and applications. A common characteristic of these solutions is operating on temporal and/or spatiotemporal data that is often in the form of data streams. Motivated by two important classes of applications, *health monitoring* and *contact tracing*, this dissertation optimizes the spatiotemporal operators in the core of these applications. The former requires efficient data streams and/or timeseries *correlations* for monitoring temporal events, and the latter optimizes *temporal aggregation joins* for spatiotemporal data (trajectories). The broader contributions of this dissertation are two novel frameworks that offer effective implementations of such applications, demonstrated experimentally with real and synthetic data.

In the context of health monitoring (e.g., server farms), we develop the Detection of Correlated Data Streams (DCS) framework. It is a real-time monitoring framework of large volumes of data streams that are produced at high velocity. Typically, pairs of most recent data streams need to be correlated within a specified delay target in order for their analysis to lead to actionable results. We address this need by: (i) segmenting data streams into *micro-batches*; and (ii) leveraging *incremental sliding window computation*, *priority scheduling*, and *caching techniques*, to avoid unnecessary re-computations and I/O. Furthermore, we devise and evaluate exploration strategies that effectively steer the processing of data stream correlations based on the monitoring objective.

In the context of contact tracing (CT), we propose the Privately Detecting Indoors Exposure Risk (PriDIER) distributed framework for detecting indoor contacts between two individuals and measuring the individual risk of infection for respiratory transmitted diseases (e.g., COVID-19). PriDIER carries out the CT queries locally on the individual users' devices to protect their privacy and utilizes a data movement protocol to achieve scalability and reduce energy consumption at the users' devices. In realizing PriDIER we develop *e-Racoon*, a novel in-memory structure to optimize temporal aggregation joins for trajectories. The *e-Raccon* structure enables efficient trajectory joins with the duration of contacts cumulatively between an individual and a single other individual, while considering the exposure across other users.

Table of Contents

List of Tables

List of Figures

Preface

I dedicate this dissertation to my family - my mother Modhi Abalkhail, my father Abdullah Alseghayer, and my brother Azzam Alseghayer. Without their support I would not have been able to finish this eventful and rewarding journey. Your belief in me, support, humor, and positiveness kept me going through the path until the end. This dissertation is dedicated for you.

Above all, I would like to thank my academic advisor Panos K. Chrysanthis for his admirable guidance and support during my PhD years. His incremental step-wise research style, holistic view of research problems, and story telling talent have shaped me to be the scientific researcher I am. I also would like to thank Constantinos Costa for his technical support, dedication to shape this work, and high availability for feedback. I would like to thank the rest of the committee members for their valuable feedback and support.

I would like also to thank my colleagues and co-authors of several papers Daniel Mossé and Daniel Petrov. Working with you has been a pleasure and a very rewarding experience. Moreover, I would like to extend my gratitude to the rest of the Advanced Data Management Technologies Lab members and my colleagues and friends at PITT that I have known during this journey, for you have been available for feedback and personal support when I needed it — (in no particular order) Luís Oliveira, Matt Barbosa, Anatoli Shein, Ekaterina Dimitrova, Brian Nixon, Salim Malakouti, Mohammad Mofrad, Kenrik Fernandis, Nathan Ong, Vineet Raghu, Nikos Katsipoulakis, Judicael Briand Djoko, Xiaoyu Ge, Xiaozhong Zhang, Amanda Crawford, Evangelos Karageorgos, Vasilis Sarris, Kenrick Fernandez, Alireza Samadian, Mahbaneh Eshaghzadeh, Zuha Agha, and Constance Clive.

To my special friends who made my journey smoother, and were there with all kinds of support, I am thankful for your presence in my life — (in no particular order) Abdullah Albarrak, Wail Alkowaileet, Yazid Al-Ismail, Mohammed Alotaibi, Luai Hasnawi, Tariq Alturkistani, Mohammed Altamimi, Amir Malki, Joon Cheol Bai, Leah Patgorski, Alexis Huet, Michel Baratin, Leila Harvard, Dave Cherry, Eleonor Ong, Stephanie Rose, David Brumble, Peter Veldkamp, James Conway, and the amazing members of the Pittsburgh

Squash Federation that has kept me motivated to stay healthy and fit during my studies.

It has been a pleasure meeting many friends from all over the world who made Pittsburgh a home for me — (in no particular order) Ally Ricarte, Jason Sichi, Christina Scenna, George Mizak, Eric Ruka, Marcela Gomez, Pedro Bustamante, Ravi Kelaiya, Raksha Koirala, Rocky Medure, Kitty Correal, Tania Clover, David Lampenfeld, Liam Lampenfeld, Chris Kramer, and Dave Cohen.

Last but not least, I would like to thank all the undergraduate and graduate students that I have taught during my journey at PITT. You have been the inspiration and motivation to keep learning, exploring, and loving my field. Your questions, curiosity, and dedication to gain knowledge has inspired me to be a better educator. I have learned from you as much as you have learned from me (and more).

"People don't care how much you know until they know how much you care" — Theodore Roosevelt

"It is impossible for a man to learn what he thinks he already knows" — Epictetus

1.0 Introduction

1.1 Motivation

The consumption (purchase and use) of sensor and smart devices has been increasing in the last decade, and the trend shows that it will keep increasing [51]. Technologies and solutions that leverage those types of devices exhibit the same pattern of increasing usage. One factor is the decrease in hardware components' prices that those devices consist of. Another factor is the high value (e.g., higher profits and convenience) that such solutions provide to individuals and organizations.

Organizations (e.g., commercial, health, and government) that utilize such kinds of solutions currently base their operational and business decisions on data analytics in order to stay in competition. Towards this, they deploy a variety of dashboards, monitoring, and tracking applications to explore and analyze large volumes of data streams looking for valuable insights and interesting events. Data streams that are produced at high velocity typically represent *timeseries* of raw measures or *spatiotemporal* data of moving objects (i.e., *trajectories*). Timeseries and spatiotemporal data share an inherent temporal ordering, while spatiotemporal data has an additional spatial ordering. This difference, i.e., the spatial dimension, leads to different challenges in managing and optimizing it.

In the context of only temporally ordered data, a common method for getting a better understanding of the observed behavior conveyed in a set of data streams is to find correlations between pairs of data streams [29, 9]. The correlation operator can also be used as a source for finding similarity measures faster [38], running threshold queries [55], or reducing the size of the data, but preserving some of its characteristics [30].

In the context of trajectories, similarity and intersections are commonly used in understanding mobility behavior and mobile analytics [47, 56, 49]. Similarity and intersection operators can support the trajectory join query that finds all the pairs of points belonging to two trajectories which are similar to each other based on space distance and time distance.

1.1.1 Correlations of Temporal Ordered Data

Finding correlations in data streams (considered as timeseries) is a challenging task. This is due to the trade-off between precision and computational cost. Current methodologies approach this challenge by employing some prediction techniques [46], Discrete Fourier Transform approximations [58, 10], or using clustering and Markov chain modeling [26]. All those approaches have their limitations, whether due to lack of absolute precision as a result of using approximations or predictions, or due to the usage of computationally expensive operations. Other approaches address this challenge by indexing the data series focusing on alleviating the computational cost [59, 24, 18]. Predominantly the users are looking for pairs of (positively or negatively) correlated data streams over a short period of time. The high number of data streams implies an even bigger number of pairs. To illustrate this challenge, consider the following monitoring application.

Example 1: *Consider a data center, operating 10,000 computers, which hosts an order of magnitude more virtual servers. A monitoring system keeps track of 20 different counters per computer (for CPU core temperature, power supply voltage, memory and network utilization, etc.). Each computer reports its counters to the monitoring system every 60 seconds. Each batch of reported data contains 12 consecutive measurements (taken 5 seconds apart). The Operations team can detect problematic servers in a timely manner and identify higher order dependencies by finding deviations from the average load per computer and negatively correlated pairs of windows of data streams as the data arrives.*

The total number of counters (i.e., data streams), which the IT specialists should analyze is 10,000 $\times$ 20 = 200,000. As every computer reports its counters once every 60 seconds, a time frame of 5 minutes will contain 60 measurements per counter. This means that there will be a total of 60 $\times$ 200,000 = 12,000,000 numbers generated every five minutes of data center uptime. This gives us almost 20 billion pairs of different data streams.

Clearly, traversing the data and calculating the correlation with full precisions is computationally expensive and induces significant delay in the production of results. The time to fully explore completely all pairs may be prohibitively long, and the challenge is exacerbated when the demand is for answers is in *real-time* and for a large set of *live* data streams.

1.1.2 Aggregate Spatiotemporal Joins of Trajectory Data

A complete spatiotemporal database systems that provide efficient execution of spatial and temporal processing of operators and analytics is still lacking [35]. Hence, few spatiotemporal systems and many access structures were proposed [1, 42, 23] to process and optimize specific spatial and temporal operators and queries used in critical applications. As mentioned above, the spatiotemporal join query is essential in mobility analytics, where two sets of trajectories are being checked for intersection in spatial and/or temporal dimension for trajectory data. The processing of such query is computationally challenging when the number of trajectories in each set is higher, since this implies higher number of pairs to be checked. To demonstrate this challenge, consider the following contact tracing application.

Example 2: *Consider the contact tracing application for COVID-19 disease, where the period of contact tracing is four previous days from the time symptoms appear, and contacts are within 6 feet with no masks. Assume the system traces 10,000 users equipped with mobile devices that stream their trajectory points on a second basis to a centralized node, where all trajectories are stored. That central node stores the users' trajectories in a relational database where the trajectory of each individual user*$_i$ *is stored in a table with the schema USER_i (timestamp, x, y), and finds the total duration of contact between two users (e.g., USER_1 and USER_2) by executing the SQL query:*

```
SELECT COUNT(*) AS duration
FROM USER_1 JOIN USER_2 ON (USER_1.timestamp=USER_2.timestamp AND
     SQRT(SQUARE(USER_1.x - USER_2.x) + SQUARE(USER_1.y - USER_2.y)) <= 6);
```

The timestamp attribute resembles the second at which a user occupied the location (x, y) in the Euclidean 2D space with the unit in the axises being in feet. The COUNT() aggregate function can help finding the number of seconds both users have intersected in time and space.

The 10,000 users result in 49,995,000 pairs of trajectories. The spatiotemporal intersection include comparing the attributes timestamp, x and y. Assuming the cardinality of each *table is the number of seconds in the trajectory, storing a trajectory over the duration of four days will have 345,600 rows. The naïve approach in comparing the rows for spatiotemporal intersection is to compare each row in a trajectory with every other row in the other trajectory as in the Nested Loop Join. Each pair of trajectories will result in having 119,439,360,000 row comparisons. This gives us about 5.97* $\times 10^{18}$ *(49,995,000* $\times$ *119,439,360,000) number of comparisons to fully process the 10,000 trajectories spatiotemporal intersections.*

Clearly, processing a large number of trajectory pairs for calculating the duration of the multiple contacts, and the cumulative viral exposure by different contacts from different sources is prohibitively costly. Even if a traditional indexing of one dimension (spatial or temporal) is employed to optimize computations, processing the other dimension basically without indexing is still costly. Partitioning the trajectory data to be processed in parallel by multiple CPU cores or locally on mobile devices may alleviate the computational cost further, but streaming users' data from their mobile devices poses a potential risk of violating their privacy. Preserving user privacy exacerbates the challenging task of computing the cumulative durations of contacts of all pairs of trajectories from multiple sources.

1.2 Hypotheses, Objective & Approach

Motivated by the two important classes of applications that are described above, namely health monitoring and contact tracing, this dissertation optimizes the spatiotemporal operators/queries in the core of these applications. Specifically, the objective is to optimize and scale the *correlations of temporal ordered data* (Aim 1), and *aggregate spatiotemporal joins of trajectory data* (Aim 2).

Aim 1: Our observation is that analyzing and correlating pairs of temporally ordered data streams is a computationally challenging task due to the prohibitively long time required to fully explore them. The high volume of pairs of streams and the high velocity of each stream render the traversal and correlation of streams high in computational cost. Moreover, the real-time and high response time requirements for processing such pairs of data streams further exacerbate the challenge. In addition to the high processing cost of such exploration tasks, there is a need to minimize the delay induced from I/O due to fetching data and processing it locally by intelligent data orchestration and storage management.

Hypothesis 1: *Our hypothesis is that efficient and intelligent grouping and processing of pairs of data streams is required in a way that reduces the delay of producing results, and increases the task throughput according to the task goal.*

To this end, we propose a solution that groups data streams in *micro-batches*. Those *micro-batches* group data streams in synchronized consistent layouts that yield a more effective exploration task. Moreover, we explore those *micro-batches* by employing a sliding window approach in order to control the temporal duration over which the correlation needs to be detected. To achieve that, we propose algorithms that quickly identify windows of correlated pairs of data streams according to the *Pearson Correlation Coefficient* [44]. Inspired by scheduling in systems, those algorithms employ scheduling techniques in order to prioritize the order of correlating the pairs in an attempt to first correlate the promising pairs that have a high likelihood of containing correlated windows. Furthermore, we utilize incremental computations in order to avoid re-computations [48]. This is achieved by having our proposed novel algorithm based on a utility function that incorporates the knowledge of incremental computations, scheduling, and caching information to correlate two synchronized windows of pairs of data streams. Finally, we consider cases where the exploration task *does* consider the detection of correlated pairs across *micro-batches*. For example, in some tasks, the goal is to detect as many *unique* pairs of correlated data streams as possible across two consecutive *micro-batches*, while in others, the goal is to *assure* the perpetual correlation between them. This is in an effort to alleviate the delay induced by exploring other pairs that are not of interest to the goal of exploration.

Aim 2: Our observation is that computing the spatiotemporal aggregation join query that considers the total durations of trajectories intersections in a single centralized node for all users in the system is computationally costly and a potential threat to users' privacy. As alluded above, optimizing the spatiotemporal aggregation join in one dimension (e.g., spatial) and processing the other one (e.g., temporal) may partially reduce its processing cost. Even if data partition is utilized to further reduce the processing cost there is still a need to compute the aggregate spatiotemporal join query in a privacy preserving fashion. While preserving the privacy of users, this query need to efficiently and effectively process the total durations of trajectory intersections that considers the cumulative viral exposure from *different sources*. Thus,

> **Hypothesis 2:** *Our hypothesis is that an aggregate spatiotemporal join query, such as in contact tracing, needs to be processed locally on users' mobile devices for maximizing their privacy while optimizing its processing through partitioning and indexing.*

To this end, we propose a new *temporal aggregation join query* that calculates the total duration from multiple sources, which is required to detect respiratory viral exposure risk of individuals locally on their mobile devices while preserving users' privacy. We propose to optimize the processing of this new temporal aggregation join query by devising a novel in-memory structure that is formed by spatial indexing in conjunction with interval trees [39, 11, 45] that encodes locations and times of contacts. In addition, we propose an energy efficient data movement protocol among users' devices that minimizes trajectory data exchanges and reduces power consumption on users' mobile devices based on their privacy requirements.

1.3 Summary of Contributions

The broader contributions of this dissertation are two novel frameworks that offer effective implementations of health monitoring and contact tracing applications, demonstrated experimentally with real and synthetic data.

In particular, we make the following contributions.

- Towards our Aim 1, we develop our ***DCS*** *(Detection of Correlated Data Streams)* framework that intelligently schedule correlating pairs of data streams within a *micro-batch* and across *micro-batches* [4, 5]. Within ***DCS***, we:
  - Develop two primary baseline algorithms that are utilized by our ***DCS*** framework, the first is ***iBRAID-DCS***, which explores the pairs in round robin fashion, and the

second is ***PriCe-DCS***, which uses a priority function based on historical success rate, cost and PCC to schedule the pairs correlation task [41, 5].
  - Devise nine different policies that our novel ***PriCe-DCS*** algorithm can employ when analyzing consecutive *micro-batches.* These policies can increase the efficiency when detecting correlated live data streams and/or address different exploration requirements. By appropriately tuning the parameters of the utility function, the different policies exhibit different *detection-recall, overlapping-recall*, and *diversity* results [3, 5].

- Towards our Aim 2, we develop a distributed framework for processing contact tracing, called ***PriDIER*** *(Privately Detecting Indoors Exposure Risk)*, that measures exposure risk across users in an effective and privacy-preserving manner. Within ***PriDIER***, we
  - Introduce two communication protocols, a *Push-based* and a *Pull-based* for trajectory dissemination within the ***PriDIER*** framework, that minimize the exchanged data and reduce the power consumption based on privacy requirements at users' mobile devices; and
  - Develop and evaluate analytically and experimentally ***e-Racoon***, a novel in-memory access structure that weaves spatial and temporal indexing to efficiently processes ***temporal aggregation join*** queries at a users' mobile devices.

1.4 Road Map

This dissertation is structured around two optimization goals. Chapter 2 covers the temporal correlation optimization solution that tackles Aim 1, where we discuss the preliminary concepts, demonstrate the proposed algorithms, evaluate our proposed solution, and discuss the related work. Chapter 3 covers the spatiotemporal optimization solution that tackles Aim 2, where we discuss the preliminary concepts, demonstrate the contact tracing framework and its components, evaluate our proposed solution, and discuss related work. Chapter 4 contains the conclusions of the dissertation and the proposed future work.

2.0 Detection of Correlated Data Streams

In this chapter, we present our *DCS (Detection of Correlated Data Streams)* framework, presented in [4], its optimization objective, and our novel algorithms *iBRAID-DCS* and *PriCe-DCS* that implement the framework's objective to optimize temporal correlations.

In the next section, we present the *DCS* framework, and in Section 2.2 we discuss our novel algorithms that the *DCS* utilizes. We show the evaluation of the discussed work in this chapter in Section 2.3. In Section 2.4 we discuss the related work to this part of the thesis and conclude in Section 2.5.

2.1 The DCS Framework

Without loss of generality, we consider a (monitoring) system that receives data from n data streams. Each data point in a data stream is a tuple t consisting of a timestamp ts and a numeric value val ($t = (ts, val)$). The timestamp captures the moment in time when the tuple was produced.

The data is produced at high velocity. The different streams produce the consecutive tuples at the same rate, and they are all synchronized. However, there are techniques to determine missing values [20, 43, 50] and to synchronize data which arrives at different rates [7, 16, 31], but they are beyond the scope of this dissertation.

The real-time analytical processing is performed in *micro-batches.*

Definition 1. *Micro-batch: A micro-batch is a group of synchronized tuple subsequences over a set of data streams defined by a timestamp* interval I.

Each micro-batch, whether of the same or different data streams, is of the same size, i.e., contains the same number of tuples with consecutive timestamps within the interval. The inter-arrival time of two consecutive micro-batches specifies the maximum computational time for processing a micro-batch.

Definition 2. *Inter-arrival Time: Starting at time δ, the inter-arrival time is the* delay target *or* deadline *d plus δ by which the last result can be produced while analyzing a micro-batch.*

In real-time processing, ideally, the deadline d is equal to the interval plus δ ($d = I + \delta$) so that there is no delay gap in processing between two consecutive micro-batches. However, it is expected to be a bit longer due to various overheads in the system, including any pre-processing of micro-batches.

The framework focuses on analytical processing that finds correlated data streams in real-time using the Pearson Correlation Coefficient (PCC) as a correlation metric for pairs of sliding windows of data streams.

Definition 3. *Pearson Correlation Coefficient (PCC): Given two numeric data streams x and y of equal length m, the PCC is calculated with the following formula:*

$$corr(x, y) = \sum_{i=1}^{m} \frac{(x_i - \mu_x)(y_i - \mu_y)}{\sigma_x \sigma_y} \tag{1}$$

where μ_x is the average (or mean) of the values of x, μ_y is the mean of the values of y, σ_x and σ_y are the standard deviations of the values of x and y, respectively.

Definition 4. *Two sliding windows of the same range w with a slide of 1 are correlated when the PCC is more than a given threshold τ ($PCC \geq \tau$).*

Definition 5. *A pair of data streams in a micro-batch is correlated when it contains at least 'A' correlated sliding windows with threshold τ.*

The windows, which meet the criterion, may be consecutive or stratified over the interval defining a micro-batch.

The formalization of the algorithmic problem that the DCS framework solves is as follows:

Problem: *Given a micro-batch B of a set of data streams DS that arrives at time δ with an arrival interval I, perfectly synchronized and with no missing tuples, and a deadline $d = I+\delta$, detect the number of correlated pairs of data streams, each of which has A correlated sliding windows, not necessarily consecutive, with a PCC threshold of τ, by the deadline $I + \delta$.*

The optimum solution will be when the number of identified correlated pairs in a micro-batch are equal to the actual total number of correlated pairs. Hence, the optimization goal in DCS is to maximize the number of identified pairs by a deadline. Formally, the ratio of the number of detected correlated pairs to the total number of correlated pairs is close to 1 and the metric is defined as:

$$Detection\text{-}Recall = \frac{\#\ identified\ correlated\ pairs}{Actual\ \#\ correlated\ pairs} \tag{2}$$

2.2 The DCS Algorithms

In this section, we discuss the two primary baseline algorithms that are utilized by our ***DCS*** framework.

2.2.1 iBRAID Algorithm

The first algorithm is what we call *iBRAID-DCS* and it is an enhancement over the work BRAID [44], where the *PCC* can be calculated by computing five sufficient statistics—sum of the values of tuples in each window, the sum of the squares of the values in the tuples of each window, and the inner cross-product of the values in tuples of the two windows, for which the correlation is calculated.

Sufficient Statistics The sum ($sumx$) and the sum of the square of the tuples ($sumxx$) of a window of length m of a data stream x, and corresponding inner product ($sumprodxy$) are denoted as

$$sumx = \sum_{i=1}^{m} x_i \qquad sumxx = \sum_{i=1}^{m} x_i^2 \qquad sumprodxy = \sum_{i=1}^{m} x_i y_i$$

The covariance of the two data streams x and y is

$$cov = sumprodxy - \frac{sumx \times sumy}{m}$$

and the variance of the window can be calculated as according to the following formula

$$varx = sumxx - \frac{(sumx)^2}{m}$$

Similarly, the variance for data stream y will be denoted $vary$. Then PCC can be calculated, applying the following formula

$$corr(x, y) = \frac{cov}{\sqrt{varx \times vary}}$$

The sufficient statistics can be computed either from scratch or incrementally each time a pair of data streams is explored by a new tuple. In the case of incremental calculation, the sums stored in memory are incremented by the new values and decremented by the values that are not part of the windows anymore. The same operations are performed for the sums of the squares and the inner cross products, using the respective tuples.

Algorithm With that in mind, *iBRAID-DCS* is a round-robin scanning algorithm that uses the incremental computation of PCC. It analyzes the pairs of data streams in a micro-batch sequentially, starting from the first tuple for all data streams. It calculates the sufficient statistics that are needed to calculate the PCC efficiently (i.e., single pass over tuples). Next, it calculates the PCC for the first tuple for all pairs of windows. Once this is done, the windows are slid further by one tuple, the sufficient statistics are updated incrementally—the first tuple is expired/subtracted from them, and the new tuple is added. The PCC is calculated again for all pairs. Then, it keeps analyzing all data streams by a single tuple, augmenting the sufficient statistics incrementally, and recalculating the PCC. This is done until the whole micro-batch is analyzed.

iBRAID-DCS has four key advantages: (1) Accurate, (2) easy to implement, (3) does not cause “starvation” among the pairs, and (4) reduces the computations by half due to the usage of the sufficient statistics. *iBRAID-DCS* is experimentally shown to perform well for data streams whose data is uniformly distributed and for low correlation thresholds ($\tau < 0.5$).

2.2.2 PriCe Algorithm

Algorithm Our second algorithm, called *PriCe-DCS*, is a scanning algorithm that uses a utility function to analyze the pairs of windows while reusing partial *PCC* computations. It analyzes the most promising pair first, which is the one with the highest utility function value:

$$Pr = PCC * (M/totalExp)/C \tag{3}$$

where *PCC* is the most recently calculated Pearson Correlation Coefficient for a pair of sliding windows that belong to the same pair of data streams, M is the number of correlated sliding windows found in the corresponding pair of data streams so far, *totalExp* is the total number of analyzed pairs of sliding windows, and C is the cost of analyzing a pair of sliding windows in terms of number of computations (i.e., the number of operations needed to calculate the sufficient statistics for a pair of sliding windows). The default values are $PCC = 1$, $M = 0$, $totalExp = 1$, and $C = 1$.

Early termination happens when the A criterion of the number of correlated windows is reached for a pair, and pruning happens according to the following condition:

$$(A - correlatedWindows) > (I - slidingWindowPosition)$$

where *correlatedWindows* are the total number of windows that are correlated in a pair of streams according to PCC τ, and *slidingWindowPosition* is the pair's analysis location in the interval. Recall, I is the interval of the data streams.

Policies When the very first micro-batch arrives at the system, the system has no prior knowledge about any correlated pairs of streams. However, this is not the case after the analysis of any micro-batch that produces a set of correlated pairs of data streams. This raises the question of how to exploit the results of past micro-batch analyses, such as, picking the first pair in a new micro-batch to analyze. This question has a major impact on *PriCe-DCS*'s behavior in supporting *exploration, exploitation* or *fairness*, and its answer determines the initialization of the parameters of *PriCe-DCS*'s utility function.

We have come up with nine policies that dictates how the *PriCe-DCS* utility function is initialized, those are:

Fresh Start: When the analysis of a micro-batch starts with no prior knowledge of correlated pairs of streams, *Fresh Start* policy initializes *PriCe-DCS*'s utility function to its default values (as discussed earlier in this section). It is to be noted, however, that the very first micro-batch analysis in all approaches follows the *Fresh Start* approach

Informed: The utility function is initialized based on the results of the latest micro-batch analysis. In *Informed* starting phase, *PriCe-DCS*'s utility function is initialized to the same parameter values of the correlated pairs used by the immediately previous micro-batch. The rationale behind this policy is to keep analyzing closely those pairs that already exhibited high correlation in the previous micro-batch, potentially indicating an insight of interest.

Untouched: The focus is on the pairs that were not processed at all in the previous micro-batch due to the lack of any correlated windows (i.e., not chosen for analysis due to their low values of Pearson Correlation Coefficient) at the beginning of *PriCe-DCS*'s execution. Specifically, such pairs are jumpstarted by altering their previous number of correlated windows (i.e., the parameter that reflects this information) to have the value A. This increases their priority, preventing their starvation and giving them another chance to be analyzed in the new micro-batch. The rationale behind this policy is to allow such pairs another chance, potentially identifying different behavior, which remained undetected in the previous micro-batch.

Alternating: This policy gives the pairs that were not correlated in the previous micro-batch a chance to be explored through a hybrid round-robin fashion. In *Alternating* starting phase, alternately, a pair from those that are not correlated in the previous micro-batch, is picked and explored using *PriCe-DCS* followed by a pair from those which were correlated. When the starting phase concludes (i.e., touched all the pairs at least once), the pairs are processed with *PriCe-DCS* according to the utility function. By doing this, we hope to reduce the effect of starvation for those that were not correlated in the previous micro-batch.

X% Non-Correlated: This policy tries to achieve fairness of exploration through jumpstarting the lowest X% pairs in priority. The pair with the highest priority among those lowest X% is picked and explored. This continues until all those X% pairs are jumpstarted. Subsequently, *PriCe-DCS* carries out the exploration process as it usually does.

Decaying History: This policy regards the significance of the whole historical correlation information of a pair differently from recent micro-batches information. In the utility function, it alters the parameter *M*, which reflects the number of correlated sliding windows found for a corresponding pair, such that it becomes weighted. It gives the historical correlation information (i.e., data from micro-batches earlier than the most recent one) a weight, and then gives a higher weight to the most recent correlation information. Then, the total of both becomes the new parameter *M*. The goal behind this policy is to consider the most recent information with a higher weight along the exploration process as opposed to older ones.

Shared Stream: Its focus is on the group of pairs, that were not correlated in the previous micro-batch but share a data stream that was part of a correlated one in the preceding micro-batch. The idea in this policy is that a data stream that is correlated with another one, might be correlated with a third different stream as well. Thus, this policy picks a pair from this group of non-correlated pairs according to *PriCe-DCS* and explores it. Shared Streams starts all those pairs, and then, carries on using *PriCe-DCS*.

X% Probing: This policy explores the first few windows for all the pairs in a round-robin fashion. This is done to set the utility function with actual current values instead of artificial hand-crafted ones. After those few windows, *PriCe-DCS* kicks in and continues the exploration process using the utility function that had its parameters filled with actual data through the probing process. The rational behind this policy is to take advantage of the good properties of *iBRAID-DCS* during the starting phase. In some respect, this policy is a hybrid of *iBRAID-DCS* and *PriCe-DCS*.

P-Alternating: This policy mimics multilevel queue scheduling, whereby the pairs are explored in a round-robin fashion between two groups. Those are the previously correlated pairs and the non-correlated ones. This is to give the pairs that were not correlated in the previous micro-batch a chance to be persistently processed. It picks a pair from the non-correlated ones and processes it using *PriCe-DCS*, then, it picks a pair from those that were correlated. It does that until the end of the micro batch (i.e., not only the starting phase), and carries on in this fashion by picking the pair from each group of pairs according to the utility function. By doing this, the hope is to alleviate the effect of starvation for those pairs, that were not correlated in the previous micro-batch in a persistent way, for they might exhibit some correlation beyond the starting phase.

2.3 Evaluation

In this section we present the evaluation of the *DCS* framework and its algorithms. Furthermore, we evaluate the nine different policies with *PriCe-DCS*, and how each policy addresses different exploration requirements.

2.3.1 Testbed

Infrastructure We implemented all the discussed strategies in C++11. We ran the experiments on a computer with 2 Intel CPUs, running at 2.66GHz, and 96GB of RAM memory. The operating system used was CentOS 6.5 and the compiler was GCC version 4.8.2.

Dataset *Yahoo Finance Historical Data* [25]: The dataset we have used in our experiments consists of 318 data streams. Those reflect the trading of 53 companies on the NYSE for the last 28 years. This gives us a total of 50,403 different pairs to analyze. The data granularity is a day, which includes the price of the stock of the company at opening, the price at the end of the day (closing), the highest price for the day, the lowest price for the day, the amount of shares traded that day, and the adjusted close (calculated according to the standards of the CRSP, Center for Research in Security Prices). The length of each data stream is about 7,100 tuples. Those tuples are divided into micro-batches.

Algorithms We compared a baseline algorithm *Random* against our two algorithms *iBRAID* and *PriCe*, along with their *DCS* variants *Random-DCS*, *iBRAID-DCS*, and *PriCe-DCS*.

Also, we studied the nine different policies that modify *PriCe-DCS* default behavior.

Metrics We evaluated the performance of the strategies in terms of *detection-recall*, *overlapping-recall*, and *diversity*. We also measure the *cost*, which is used to determine the deadlines in our experiments.

Detection-Recall: This is our detection optimization criterion (Eq. 2). It reflects how capable the strategy is in detecting correlated pairs out of the total actual correlated pairs. Thus, it is a ratio of the number of detected correlated pairs to the total number of correlated pairs.

Overlapping-Recall: We call pairs that were detected in a given micro-batch and also were detected in the preceding micro-batch as an overlapping pair. In this metric, we find the ratio of the detected overlapping pairs to the total number of overlapping pairs in a micro-batch. Note that this metric does not apply to the very first micro-batch.

Diversity: We measure how many new pairs (i.e., not seen as a result in the most recent micro-batch) are detected in each micro-batch. This is our exploration *vs* exploitation criterion.

Cost: This is our efficiency metric. We measured the deadline latency as the number of operations performed to detect correlated pairs of data streams. We used the number of operations as it provides the asymptotic efficiency of the strategies compared to one another. This does not depend on factors such as the hardware characteristics and the operating system of the computer, on which the experiments are run, nor the efficiency of the compiler. We examined how the strategies meet deadlines and how many correlated pairs they could detect under such a requirement.

Table 1: Experimental Parameters

Parameter	**Value(s)**	**Parameter**	**Value(s)**
PCC τ	[0.75, 0.90]	w	8
A	[112, 225, 450]	# data streams	72
I	900 (180 seconds)	# *micro-batches*	4

2.3.2 Experiments

We ran four experiments in total. The first two evaluate our base algorithms, and the latter two assess the ability of our proposed policies to detect and diversify the correlated pairs of data streams using *PriCe-DCS*. We conducted the experiments for two PCC threshold τ's, 75% and 90%, and for three different values of A, 112, 225 and 450. The values of A correspond to the 1/8, 1/4 and 1/2 of the micro-batch interval. The micro-batch interval is set to 900 tuples to simulate an inter-arrival time of 180 seconds, where each tuple is produced each 200 milliseconds. Finally, we experimented with three deadlines corresponding to 25%, 50%, and 75% of the total operations needed to determined all the correlated pairs in a micro-batch, i.e., achieve total *detection-recall*. The experimental parameters are summarized in Table 1.

In all experiments, we have divided the dataset into four mutually exclusive groups, and we ran our experiments on all of them, we found that the results are similar. Thus, we reported the results of one of those groups. Moreover, we did pick 10% for the *X% Non-Correlated* as a middle point between the policies *Untouched* and *Alternating*.

2.3.2.1 Experiment 1 (Figs. 1–3)

In our first experiment, we measured the execution *cost* or latency in *number of operations* of each algorithm to detect the specified number A of correlated pairs of sliding windows in four consecutive micro-batches.

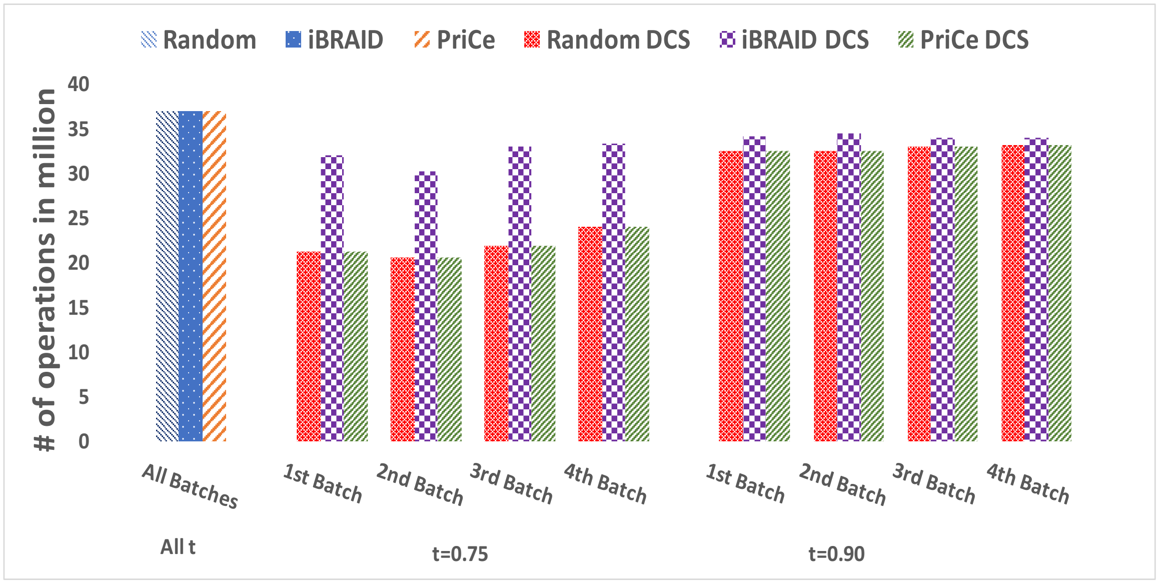

Figure 1: The cost in number of operations for 4 consecutive micro-batches ($A = 112$).

As expected, *Random*, *iBRAID*, and *PriCe* have the same number of operations due to exhaustive processing of the pairs. These do not use A for either early termination nor pruning. Thus, with fixed number of data streams, intervals, and window range, the number of operations induced by the three algorithms is identical. They always consume the same amount of operations regardless of the other parameters (i.e., PCC τ and A). The impact of *DCS* mode on all three algorithms is clearly visible in all figures.

In Figure 1, we notice that the higher the PCC τ, the more operations are executed by the algorithms. This is due to the fewer number of windows that are highly correlated according to the PCC τ. This results in higher latency in detecting them by the algorithms. On the other hand, in case of low PCC τ, we see that *DCS* terminates the analysis process early, as soon as it reaches the A criterion of number of correlated sliding windows. We also observe that *iBRAID-DCS* consistently underperforms the other *DCS* algorithms in latency. This is a consequence of the *iBRAID* scheduling scheme which leads to having a pair pruned at a late stage of the analysis.

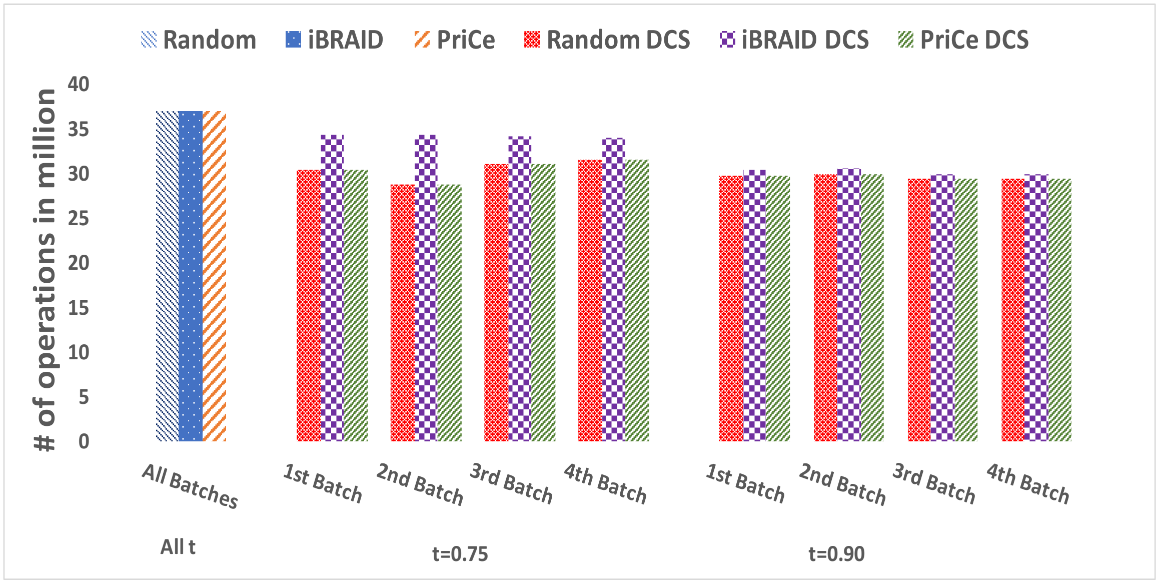

Figure 2: The cost in number of operations for 4 consecutive micro-batches ($A = 225$).

In Figure 2, we notice that *iBRAID-DCS* exhibited higher latency in the cases where PCC $\tau = 0.75$. This is counter intuitive. To clearly state the reason, we observe that the lower the A criterion, the later the pruning will occur in case of no high correlated windows were processed. With that in mind, we say that the pairs of data streams in Figure 2 in the case of PCC $\tau = 0.75$ has produced high amount of correlated windows, enough to delay the pruning towards the end, but not enough to terminate the analysis early. Therefore, the performance of *iBRAID-DCS* was lower with low PCC τ.

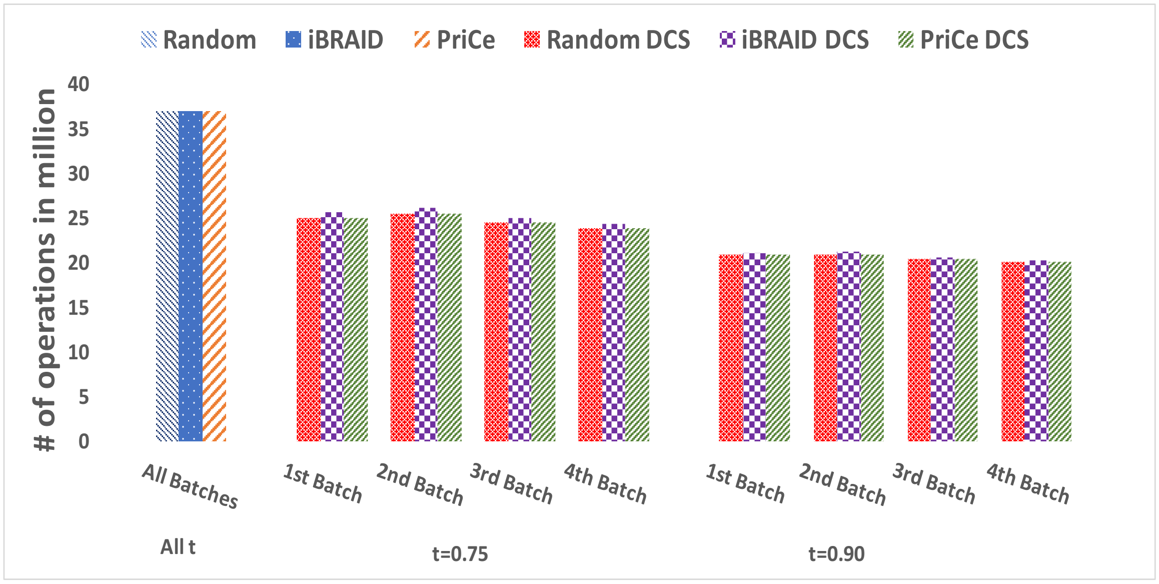

Figure 3: The cost in number of operations for 4 consecutive micro-batches ($A = 450$).

Our last explanation is also supported by our experimental results in Figure 3, which show clearly that with high A, the *DCS* mode is able to reach a better performance than low A. The reason is that with $A = 450$, if a pair encounters no correlated windows yet, it can be pruned by midway of the analysis process.

Finally, *DCS* mode of operation was able to enhance the performance of the algorithms up to 1.8 times (Fig. 3).

2.3.2.2 Experiment 2 (Figs. 4–8)

In this experiment, we studied the *Detection-Recall* of each algorithm with respect to a given deadline. We set the deadline to be 25%, 50%, and 75% of the processing duration of each interval and measured the percentage of the number of correlated pairs each algorithm was able to detect. The results are shown in Figures 4–6 for the deadline 25%, in Figure 7 for the deadline 50%, and in Figure 8 for the deadline 75%.

In general, we notice that *DCS* mode of operation in all cases for all algorithms outperforms the original algorithms. This is attributed to the pruning and early termination features of *DCS*, which allow the algorithms to analyze other pairs and detect more correlated data streams.

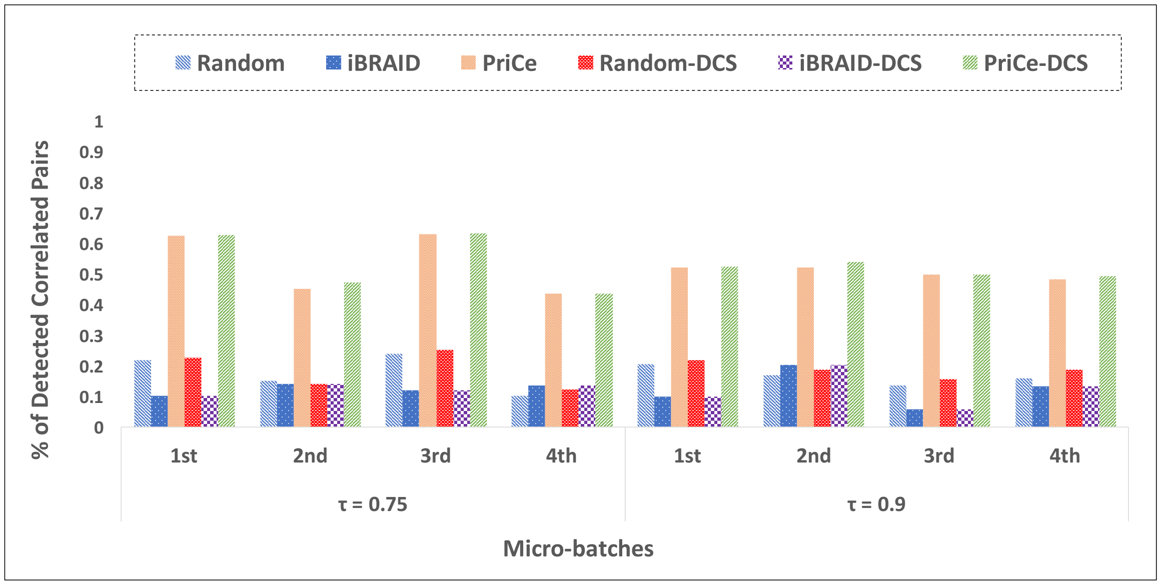

Figure 4: The % of correlated pairs of streams detected by all algorithms at 25% of the interval I ($A = 112$).

In Figure 4, we notice that *iBRAID* and *iBRAID-DCS* detected lower percentage of correlated pairs of data streams than *PriCe* and *PriCe-DCS*, while *PriCe* and *PriCe-DCS* have comparable performance. We attribute this to the fact that with a low value of A, we are expecting the required A number of correlated pairs to be detected quickly by *PriCe* and *PriCe-DCS*, while the round-robin scheduling of *iBRAID* and *iBRAID-DCS*, which process all the pairs one pair at a time, is not affected by the value of A. We also notice that *Random* performed slightly better than *Random-DCS* in the 2nd micro-batch, and this is due to the random scheduler nature that picks a pair in a random fashion. In addition, we observe that the average detection percentages for *Random* with A=112 for all PCC τ at the deadline 25% is 18% comparing to 52% for *PriCe*.

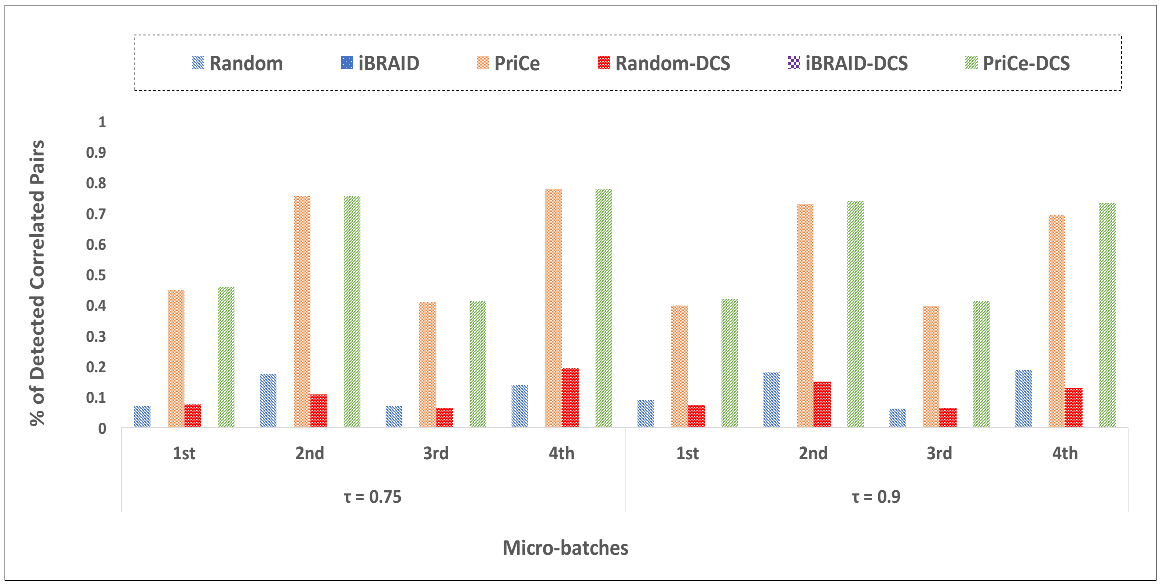

Figure 5: The % of correlated pairs of streams detected by all algorithms at 25% of the interval I ($A = 225$).

In Figure 5, we see that *PriCe* scheduling demonstrated the effectiveness of its priority function in capturing more correlated pairs at an early deadline, especially when PCC τ is high. That means, it elects the pairs to explore more intelligently than the other two algorithms. We also notice that with high value of A and early deadline, *iBRAID-DCS* fails to detect any correlated pair of data streams. With respect to *Random* and *Random-DCS*, we observe the same as in Figure 4, where *Random* performed slightly better than *Random-DCS* due to the random scheduler nature. We also note that the average detection percentage for *Random-DCS* with A=225 for all PCC τ at the deadline 25% is 12% compared to 57% for *PriCe*.

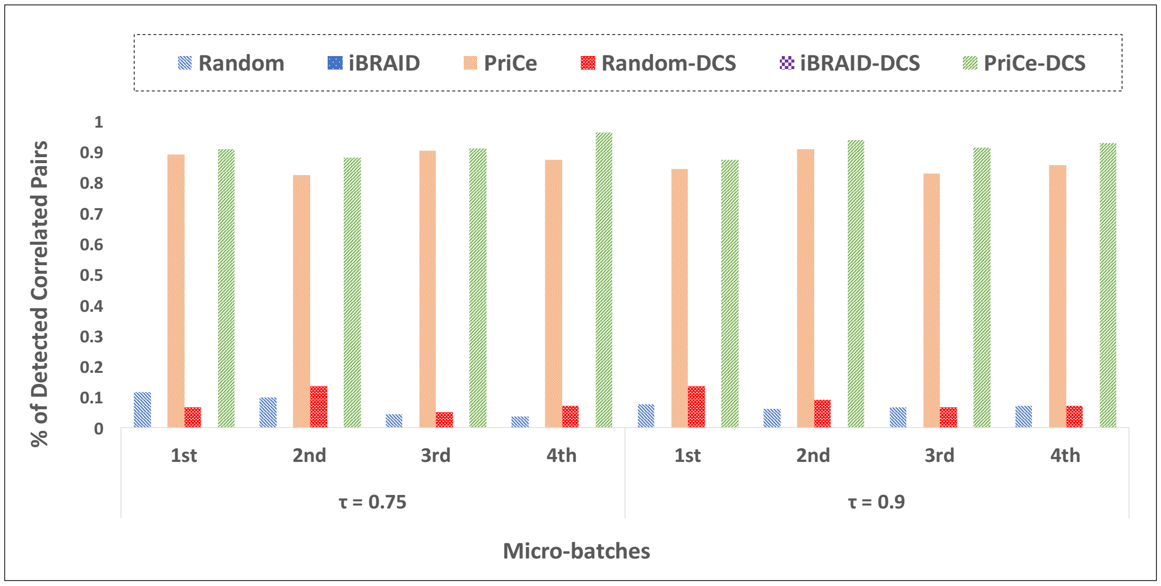

Figure 6: The % of correlated pairs of streams detected by all algorithms at 25% of the interval I ($A = 450$).

In Figure 6, the observations are as similar as they are in Figure 5, however, we realize that *PriCe-DCS* has detected more than 87% the correlated pairs of data streams, and this is due to the priority scheduling of *PriCe* and the aggressive pruning at high A. The *Random-DCS* fails to meet that, since it picks pairs in an unpredictable way, and this delays the analysis duration for each pair, hence, delaying its pruning.

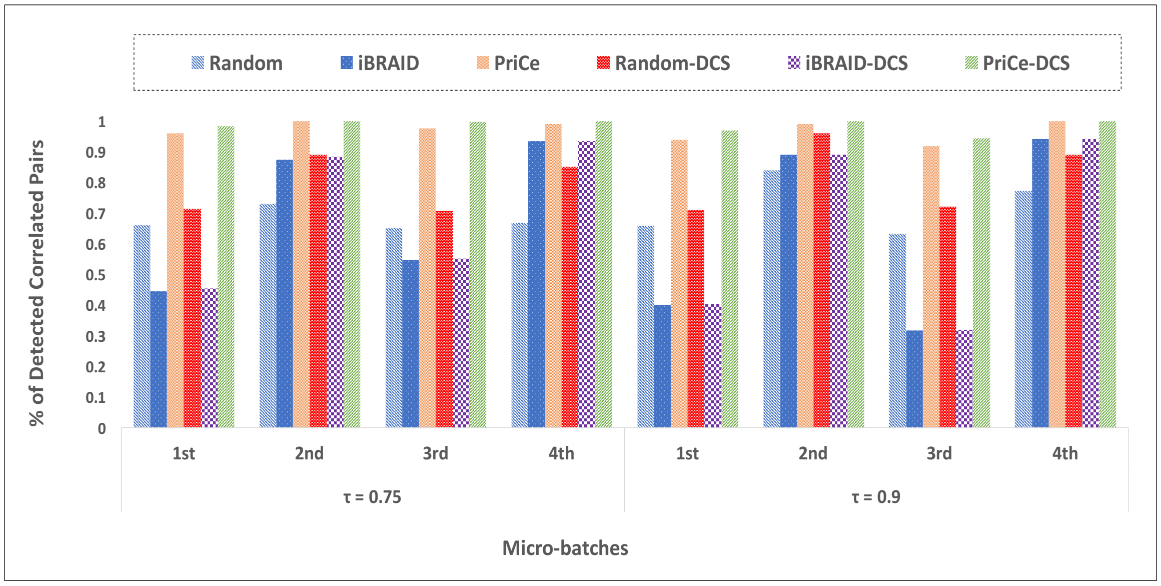

Figure 7: The % of correlated pairs of streams detected by all algorithms at 50% of the interval I ($A = 225$).

In Figure 7, we note that *PriCe-DCS* outperforms all the other algorithms. This is for the obvious reason of having more processing time to advance the sliding windows and capture more correlated windows between a pair of data streams. As a result, it reaches the criterion A ($= 225$) of declaring the pair as a correlated one quickly. For $A = 112$ and $A = 450$, our observations are similar as in Figure 7.

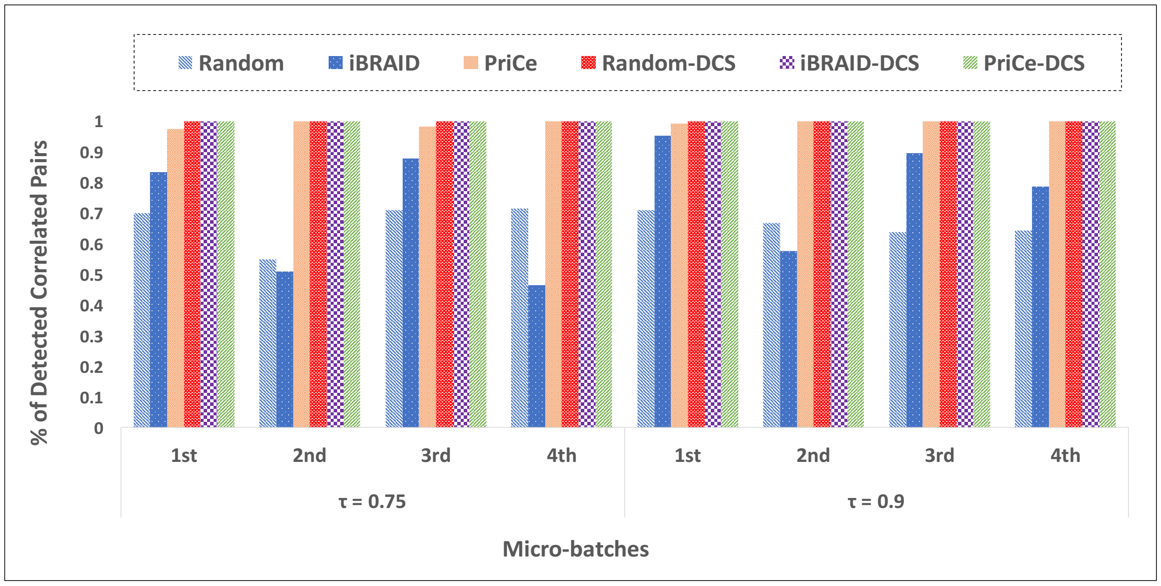

Figure 8: The % of correlated pairs of streams detected by all algorithms at 75% of the interval I ($A = 450$).

Finally, towards the end of the micro-batch analysis process, we see clearly in Figure 8 that all the algorithms are detecting pairs with an overall relatively higher percentage than the earlier deadlines, and this is expected with more time to analyze the pairs of data streams. We also observe the effect of pruning clearly on all algorithms under *DCS* mode. This is a result of the A criterion being very high, which leads to early pruning for non promising pairs of data streams. Thus, all the correlated pairs of data streams were detected earlier than the 75% deadline. The same holds for $A = 112$ and $A = 225$.

2.3.2.3 Experiment 3 (Figs. 9–10)

Our previous two experiments show that *PriCe-DCS* exhibits the best performance overall. In our third experiment, we studied the *detection-recall* of each policy with respect to a given deadline. We set the deadline to be 25%, 50%, and 75% of the processing duration of each interval and measured the percentage of the number of correlated pairs each policy was able to detect. We experimented with the values of PCC τ and A, shown in Table 1. All experiments produced similar results; thus, we report the results for the deadline 25%

and 50% only.

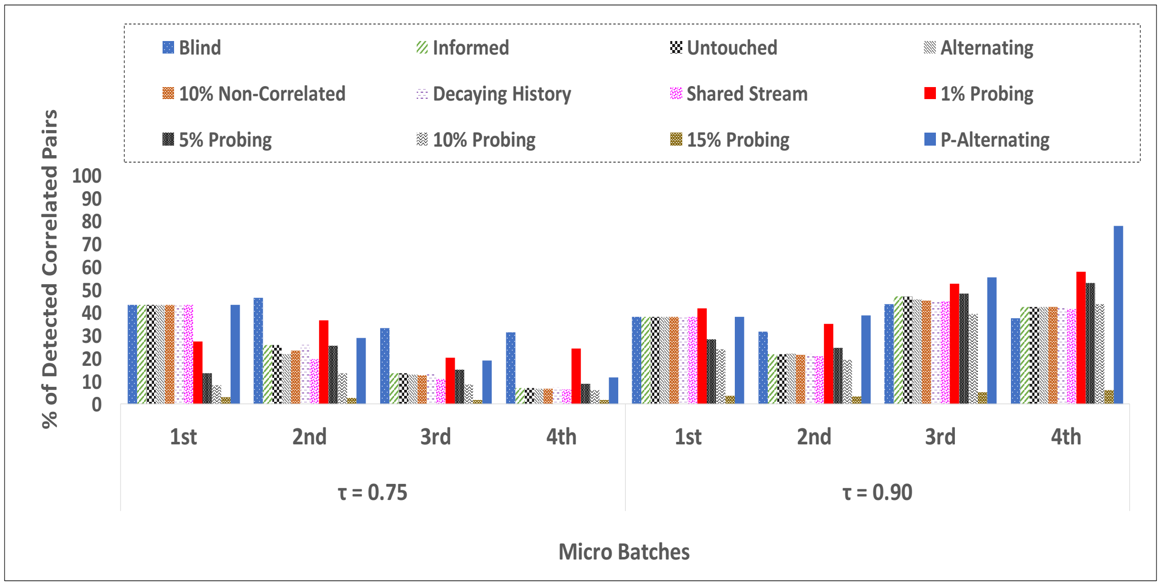

Figure 9: The % of correlated pairs of streams detected by all policies at 25% of the interval I (A = 112).

Figure 9 shows the *detection-recall* for the 25% deadline with A=112. We notice that *PriCe-DCS* with policy *P-Alternating* outperforms the rest with PCC τ = 0.90. On the other hand, the *1% Probing* outperforms the rest when PCC τ = 0.75. This is attributed to the fact that with higher PCC τ values it is highly likely to have fewer correlated windows in a given pair, and vice versa. Having that in mind, with lower PCC τ, exploring the initial 1% of a pair, can lead to an indication of whether the pair is correlated or not. On the other hand, when the pair has scarce correlated windows, it is expected that with persistent exploration of non-correlated pairs, to find new correlation discoveries. Note that the first micro-batch always starts with a *Blind* policy, as there are no historical information about the data streams.

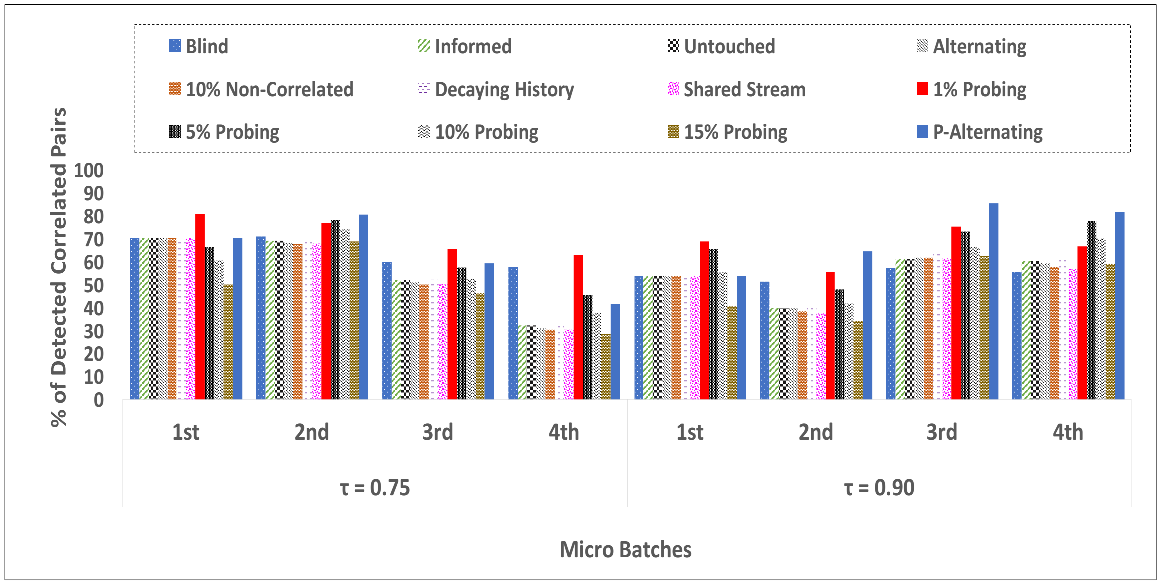

Figure 10: The % of correlated pairs of streams detected by all policies at 50% of the interval I (A = 112).

Figure 10 shows the *detection-recall* for the 50% deadline with A=112. It is clear that they exhibit similar behavior overall, except for the 2nd micro-batch, where the policy *P-Alternating* won by a negligible margin over the *X% Probing* policies.

2.3.2.4 Experiment 4 (Table 2)

In this experiment, we studied the impact of historical information on the effectiveness of detecting correlated pairs. This includes the trade-off between exploration and exploitation in the approach for detecting correlated pairs. We use the metrics *detection-recall*, *overlapping-recall*, and *diversity* to illustrate that impact.

In Table 2, we show the results for the algorithm *PriCe-DCS* with *Blind* policy as a baseline. In addition, we show the results of *PriCe-DCS* with *Informed* and *Untouched* as the most exploitative starting phase varieties. This means that they keep detecting the same pairs of data streams that they already have detected. We also show the winners from Experiment 3 (i.e., *1% Probing*, *5% Probing*, and *P-Alternating*).

Table 2: DCS Strategies Effectiveness of Detecting Correlated Pairs

		25% Deadline					50% Deadline				
	Batches	1st	2nd	3rd	4th	Avg.	1st	2nd	3rd	4th	Avg.
	Full Correlated	429	580	236	234	–	429	580	236	234	–
	Full Overlapped	—	364	215	181	–	—	364	215	181	–
PriCe Fresh Start	Correlated	163	184	103	88	–	231	298	135	130	–
	Overlapped	—	70	67	54	–	—	140	98	84	–
	Unseen Before	163	114	36	34	–	231	158	37	46	–
	Detection-Recall	0.380	0.317	0.436	0.376	0.377	0.538	0.514	0.572	0.556	0.545
	Overlap-Recall	—	0.192	0.312	0.298	**0.267**	—	0.385	0.456	0.464	**0.435**
	Diversity	1	0.620	0.350	0.386	**0.589**	1	0.530	0.274	0.354	**0.540**
PriCe Informed	Correlated	163	126	111	99	–	231	232	144	141	–
	Overlapped	—	81	99	87	–	—	144	136	118	–
	Unseen Before	163	45	12	12	–	231	88	8	23	–
	Detection-Recall	0.380	0.217	0.470	0.423	0.373	0.538	0.400	0.610	0.603	0.538
	Overlap-Recall	—	0.223	0.460	0.481	0.388	—	0.396	0.633	0.652	0.560
	Diversity	1	0.357	0.108	0.121	**0.397**	1	0.379	0.056	0.163	**0.399**
PriCe Untouched	Correlated	163	126	111	99	–	231	232	144	141	–
	Overlapped	—	81	99	87	–	—	144	136	118	–
	Unseen Before	163	45	12	12	–	231	88	8	23	–
	Detection-Recall	0.380	0.217	0.470	0.423	0.373	0.538	0.400	0.610	0.603	0.538
	Overlap-Recall	—	0.223	0.460	0.481	0.388	—	0.396	0.633	0.652	0.560
	Diversity	1	0.357	0.108	0.121	**0.397**	1	0.379	0.056	0.163	**0.399**
PriCe 1% Probing	Correlated	179	203	124	135	–	295	323	178	156	–
	Overlapped	—	123	101	90	–	—	178	129	111	–
	Unseen Before	179	80	23	45	–	295	145	49	45	–
	Detection-Recall	0.417	0.350	0.525	0.577	0.467	0.688	0.557	0.754	0.667	0.666
	Overlap-Recall	—	0.338	0.470	0.497	0.435	—	0.489	0.600	0.613	0.567
	Diversity	1	0.394	0.185	0.333	0.478	1	0.449	0.275	0.288	0.503
PriCe 5% Probing	Correlated	121	143	114	124	–	281	278	173	182	—
	Overlapped	—	110	100	96	–	—	184	145	134	—
	Unseen Before	121	33	14	28	–	281	94	28	48	—
	Detection-Recall	0.282	0.247	0.483	0.530	0.385	0.655	0.479	0.733	0.778	0.661
	Overlap-Recall	—	0.302	0.465	0.530	0.433	—	0.505	0.674	0.740	0.640
	Diversity	1	0.231	0.123	0.226	0.395	1	0.338	0.162	0.264	0.441
PriCe P-Alternating	Correlated	163	224	131	182	–	231	375	202	191	—
	Overlapped	—	108	124	130	–	—	209	187	173	—
	Unseen Before	163	116	7	52	–	231	166	15	18	—
	Detection-Recall	0.380	0.386	0.555	0.778	**0.525**	0.538	0.647	0.856	0.816	**0.714**
	Overlap-Recall	—	0.297	0.577	0.718	**0.531**	—	0.574	0.870	0.956	**0.800**
	Diversity	1	0.518	0.053	0.286	0.464	1	0.443	0.074	0.094	0.403

One can notice that the *P-Alternating PriCe-DCS* achieved the highest *detection-recall* on average for both deadlines. This is expected due to the persistent processing of the non-correlated and correlated pairs in a round-robin fashion. In addition, although *Informed PriCe-DCS* achieved high *overlapping-recall* on average, *P-Alternating* has the highest of all for both deadlines. This is to be expected because *Informed PriCe-DCS* does not alter the parameters of the utility function, instead, it carries all the information of pairs from previous micro-batches as they are, and in the case of *P-Alternating*, the persistent processing of the already correlated pairs manifests itself in the highest *overlapping-recall*.

Finally, the explorative policies (i.e., *Blind*) achieved the highest *diversity* in detecting correlated pairs than the exploitative ones (i.e., *Informed*, *Untouched*, and *P-Alternating*), in fact, they have achieved the lowest *diversity* on average. This can be explained because the exploitative policies have some kind of informative approach on how to analyze the data streams, whether this is from previous micro-batches or some other source. Thus, they will keep exploring the pairs that were already correlated in previous micro-batches.

2.3.2.5 Discussion

In our first two experiments, we found that *PriCe-DCS* outperformed all other algorithms. In the third, we found that *PriCe-DCS* with *X% Probing* is the best policy for detecting correlated live data streams for low values of PCC τ. On the contrary, *PriCe-DCS* with *P-Alternating* is the best policy for detecting correlated live data streams for high values of PCC τ. In the last experiment, we found that *P-Alternating*, *Informed* and *Untouched* policies are more suitable for exploiting the space of exploration, and for finding correlated pairs regardless of *diversity*. However, *Blind* policy does detect more diverse pairs across micro-batches along the exploration process.

2.4 Related Work

The processing of data and fast discovery of correlated data streams is tackled in two scenarios with respect to the production of data: 1) static, when the data is collected upfront and it forms the search space for finding the correlated subsequences [28, 44, 34], and 2) dynamic, when the data is processed as it is produced [46, 58, 10, 26, 33]. The former is beyond the scope of our work. In this section, we focus on the latter, and in particular the state-of-the-art of computationally cheap identification of correlated data streams.

RainMon [46] proposes a 3-stage technique to mine bursty data streams. The received signals are first decomposed, in order to obtain a smooth representation of the data. In the next stage, called summarization, the received data goes through incremental principal component analysis in an effort to outline the long-term trends in the streams and to identify anomalies, if there are any. In the last stage, named Prediction, the system forecasts trends, relying on the output from the summarization stage. In our work, we do not make predictions and use the data as it is delivered to identify correlated data streams.

A framework for identification of highly correlated pairs of data streams is also presented in StatStream [58]. One of the assumptions of the work is that only an approximation of the PCC is sufficient to identify the pairs of highly correlated data streams. Based on these assumptions, the authors proposed a twofold approach to efficiently identify the pairs of interest. They employed a computationally cheap Discrete Fourier Transformation (DFT) technique to calculate an approximation of the PCC. Furthermore, they proposed an n-dimensional grid structure, which stores the DFT statistics and PCC approximations of each stream, whereby neighboring cells reflect highly correlated streams. This is the springboard for identification of the highly correlated pairs of data streams. However, DFT is known for its poor performance on data streams, which mimic white noise, thereby the required number of DFT coefficients to precisely represent the data streams is high, which induces a significant amount of computations. Our work differs in the calculation of PCC. Furthermore, we calculate PCC incrementally over sliding windows, and our framework calculates it precisely for each pair, over each sliding window. Our studies showed that once a pair of data streams is selected as being highly correlated due to a high value of the approximated PCC, a precise calculation of the PCC is required to prove the hypothesis. This operation requires two passes on the data. Similarly to StatStream, our framework supports sliding windows on data streams. We evaluate the possibilities to extend our framework to support landmark windows and damped windows in the future.

StatStream was further improved to handle "uncooperative" data streams in [10], but it still calculates an approximation of PCC only. The proposed technique employs structured random vectors. The experimental results show that the proposed technique outperforms linear scan and the Discrete Fourier Transformations, proposed in StatStream [58]. Our framework, similarly to this work, updates the required statistics in a fixed amount of time. However, it differs in PCC calculations, whereby *DCS* calculates PCC of the pairs of data streams *precisely* for each sliding window once a pair is selected as being "promising" and avoids the need to be calculated later and at a higher cost.

Detecting similarities between data streams can be achieved through correlation identification techniques. Four different distance measures for similarity of data streams were proposed in [33]: "Autocorrelation Distance (ACD)," "Markovian Distance (MD)," "Local Distance Distribution" and "Probabilistic Local Nearest Neighbor." *ACD* is the version of similarity metric used in our work (PCC), but used for self-correlation, i.e., when a data stream is correlated to itself, whereby one of the windows starts with a lag from the other one. All discussed methods are used to find only the first nearest neighbor (1NN) of a given data stream. Our approach, however, identifies all pairs of correlated data streams and is not limited to 1NN only.

Anomaly detection over data streams can also be used as a correlation identification method. A solution presented in [26] uses the *MD* approach, listed above. Specifically, the presented solution relies on a twofold approach, whereby data streams clustering is combined with Markov chain modeling. The former identifies groups (or clusters) of similar data streams. The latter conveys a possibility for the system to identify anomalies in the data streams in each cluster. In the context of the system, anomalies are considered to be transitions in the Markov chains, which have probability below a certain predefined threshold. Our work may not only be adjusted to identify anomalies, whereby an anomaly is a pair of windows with PCC below a certain threshold, but it also provides analysts with insights about the data. This is done by employing cheap incremental computations, avoiding computationally expensive operations such as building Markovian transition matrices.

2.5 Summary

In this chapter, we presented the *DCS* framework which offers effective solutions for detecting the correlation of data streams within a micro-batch of a fixed time interval for specific analysis requirements.

Specifically, we first discussed our simple and fair *iBRAID-DCS* algorithm that explores pairs of data streams in round-robin fashion and prevents pairs starvation during the exploration task. Then, we discussed *PriCe-DCS* algorithm, which combines (1) incremental sliding-window computation of aggregates and (2) intelligent scheduling, driven by a utility function. At last, we discussed how the *DCS* framework facilitates the implementation of different policies that tune the utility function in order to meet the exploration and exploitation requirements of analysis tasks.

We implemented and evaluated nine policies that initialize/tune the utility function of *PriCe-DCS*. Other policies could potentially be specified. As opposed to the policies that address explorative objectives, such as *Fresh Start* and *X% Probing*, the policies that address an exploitative objective, such as *P-Alternating*, *Informative*, and *Untouched*, use the result of the preceding micro-batch analyses as part of the initialization of the analysis of the current micro-batch.

Our experimental evaluation using real world dataset showed that the policies that address explorative objectives (i.e., *Fresh Start* and *X% Probing*) detected more unique correlated data streams. It also revealed that for low PCC τ, *PriCe-DCS* with *X% Probing* outperformed the rest of the policies in terms of *detection-recall*, and for high PCC τ values, *PriCe-DCS* with *P-Alternating* performed the best.

3.0 Privately Detecting Indoors Exposure Risk

In this chapter we present our *PriDIER* (Privately Detecting Indoors Exposure Risk) contact tracing framework, its component, and our novel in-memory structure *e-Racoon* that implement the framework's query processor for optimizing the temporal aggregation joins.

In the next section, we present the essential definitions and preliminary concepts related to the *PriDIER* framework. In Section 3.2, we introduces the concept of contacts in contact tracing, the contact tracing (CT) framework *PriDIER*, and its components. In Section 3.3, we formally define the *temporal aggregation join* query and discuss how its processed and optimized using *e-Racoon* our novel in-memory access structure. We discuss our experimental testbed and our experimental evaluation in Section 3.4. We discuss the related work and state-of-the-art solutions in Section 3.5 and conclude the chapter in Section 3.6.

3.1 Preliminary

In this section we define some preliminary concepts, and in Table 3 we provide the notations used in this work.

Definition 1. *Trajectory: A trajectory* $T = \{p_1, p_2, ...p_n\}$ *is a finite chronologically ordered set of points that belongs to a single person using a mobile device. A trajectory point* $p_i = (x_i, y_i, t_i)$*, where* x_i *and* y_i *are Euclidean coordinates and* t_i *is the timestamp at which the point* p_i *was sampled by the mobile device.*

Table 3: Notations

Notation	**Definition**
MO	Moving object
CA	Trusted central authority
S	MO sampling rate
T_l	Locally stored trajectory at the MO
T_q	Externally received query trajectory
p_i	ith point in a trajectory
x_i	ith point x coordinate
y_i	ith point y coordinate
ts_i	ith point sampling timestamp
Z_i	ith point spatial zone
W_t	A time window in seconds
σ	Disease transmission window prior symptoms
ϵ	Close contact spatial constraint
δ	Close contact temporal constraint
α	# contact segments in a pair of trajectories
β	Total number of points over contact segments
γ	# stabbed contact points (always equal to α)

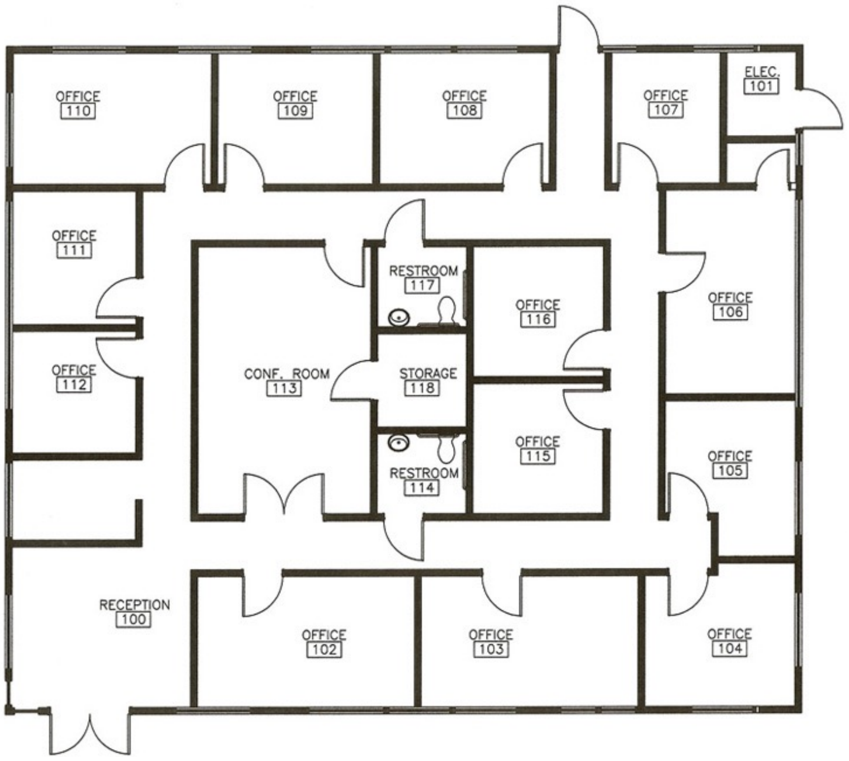

Figure 11: An example of a static segmentation of a floor into zones, where each office is a zone and the hallways are all a single zone.

Definition 2. *Zone: A zone Z_i is a static area that is defined over the Euclidean space. Zones could be equally sized cells in a grid or could be different in size based on the physical layout (e.g., offices, rooms). A zone has neighboring zones, a collection of zones form a floor (Figure 11), and a group of floors are contained in a building.*

It should be noted that each point p_i in a trajectory is contained within a specific zone Z_i, i.e., $\{(Z_i, ts_i)\}$.

Definition 3. *Close Contact: Close contact occurs when two people are spatially collocated within ϵ distance (e.g., 6 feet, or ≈1.8 meters in case of COVID-19), at any time starting σ days before the respiratory transmitted disease symptoms began (e.g., two days in case of COVID-19), and for at least δ seconds (e.g., 15 minutes, or 900 seconds in case of COVID-19) cumulatively within a window W_t (e.g., 24 hours in case of COVID-19).*

Definition 4. *Temporal Enveloping: Temporal enveloping is the act of extending a specific timestamp to a time interval* $[t_l, t_h]$*, where* t_l *is the lower timestamp and* t_h *is the higher timestamp. Extending time is for the purpose of accounting for residual particles in air after a person leaves the area (indirect contact). In other words, if a person existed in a specific zone for a duration of time (an interval)* $[t_l, t_h]$*, then enveloping (extending) that interval with* x *duration results in the temporally enveloped interval* $[t_l - x, t_h + x]$*.*

The correctness of our temporal enveloping scheme can be proved as follows:

Proof. Assuming that after x time of infected MO leaving a certain location, and assuming that after enveloping a specific time stamp and converting it into an interval, there will be infectious particles in the air. If the enveloping duration is y, and an infected MO existed for the duration of $[t, t + x]$, then the space will be temporally considered occupied for the enveloped interval $[t - y, t + x + y]$. This implies that any MO who existed in the same space y time before it was occupied, and y time after it was occupied are safe from infectious particles. This leads to a contradiction. □

Definition 5. *Spatial Enveloping: Spatial enveloping is the act of considering the surrounding areas of a specific exact point in space to account for inaccuracies. We achieve enveloping in this work by having zones accounting for areas that are large enough (at least* ϵ *square feet) to be self-contained (e.g., meeting area, elevator).*

Figure 12 shows an example of the effect of spatial enveloping. Notice that MO_1 enters the floor and leaves without being in contact with any other MOs. However, even though MO_2 does not cross path directly with $MO_{infected}$, they cross path at the enveloped space of $MO_{infected}$, which we consider as a contact.

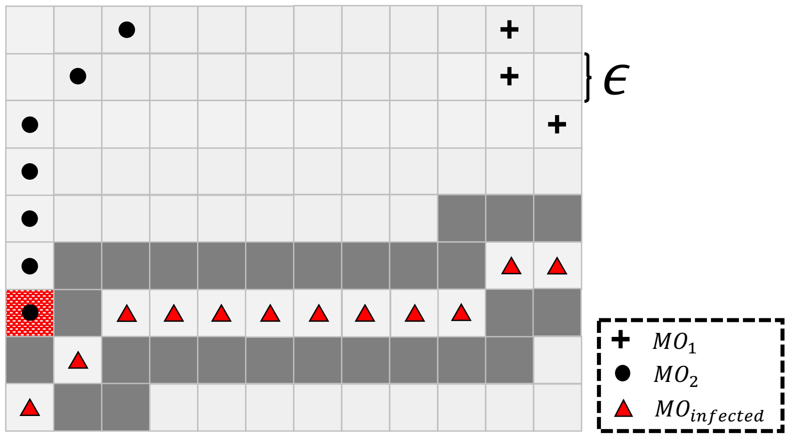

Figure 12: Example of three different MOs interacting in a space where each zones has an area of ϵ^2.

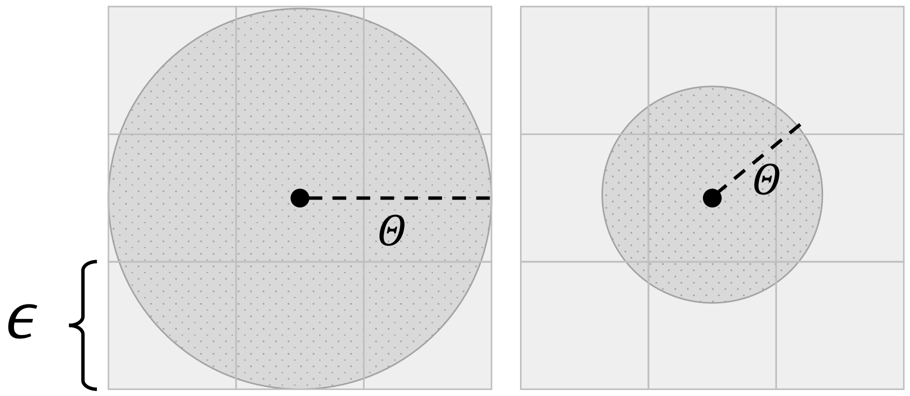

Figure 13: Two different MOs with different average sensors inaccuracies radius (θ), where a MO is located at the center of a zone and surrounded by enveloping zones.

Assuming that the average moving object's location sensor error is θ, we pick that average error to be the radius of the circle centered at the moving object's sampling location (Figure 12). We define θ in terms of ϵ as $\theta = x\epsilon$. To ensure inclusive enveloping, we need to include all the zones (cells in the figure) that overlaps with the circle centered at the MO. In Figure 13, we show two examples with different inaccuracy values for different localization devices (θ values).

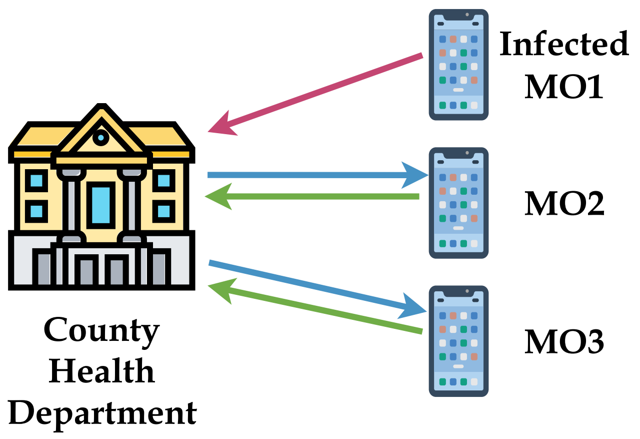

Figure 14: CA and MOs interacting to achieve exposure measurement and close contact detection.

The correctness of our spatial enveloping scheme can be proved as follows:

Proof. Assuming that the radius of localization devices error is θ, which equals $x\epsilon$. This implies that 1) the sensed point is contained within the circle centered at the moving object and has the radius is $x\epsilon$. 2) The square zone (or group of square zones) that perfectly contains the circle (circle fit within and tangent to four sides of the square) is larger in area than the contained circle. That is, minimum the square with the area $(2x\epsilon)^2$ that is centered at the moving object. Now, assuming that the sampled point error renders the point outside the containing square with the area $(2x\epsilon)^2$, that is centered at the moving object, also knowing that $\theta = x\epsilon$, leads to a contradiction. □

Definition 6. Moving Object *(MO) is a user who is possessing a localization device (e.g., cellphone with a GPS sensor), and has the ability to change their location at any time (i.e., move around in the system's spatial map). MOs sample locations and store them locally in a trajectory data format:* $\{.., (Z_i, ts_i), (Z_j, ts_j), ...\}$

Definition 7. Trusted Central Authority *(CA) is a* trusted agent *(e.g., a county health department) that all MOs trust to stream their trajectory data to, and request infected tra-*

jectories from, in an anonymized and encrypted fashion (Figure 14). The role of CA is to keep track of infected MOs and informing all MOs in the system with newly infected MOs in order to conduct the contact tracing *tasks and/or the* exposure measuring *task.*

3.2 Contact Tracing Framework

Our framework, called *PriDIER* (Privately Detecting Indoors Exposure Risk), is an infrastructure of participatory users (MOs) storing their trajectories locally at their end and a CA that manages infected trajectories. Once an MO becomes infected, it notifies and submits its (infected) trajectories to the CA—an infected MO is a user who had a close contact with an infected user, has an exposure risk, or medically tests positive for a respiratory transmitted disease. The CA informs the MOs of infected trajectories, enabling them to carry out contact tracing locally to protect their privacy.

Upon receiving an infected trajectory, an MO executes a *temporal aggregation join* on the received trajectory and its own to determine if any close contact has occurred and the total risk exposure duration. The total duration of exposure could be applied to the HealthDist Quanta Calculator [14], and users can determine their potential risk of exposure.

The formalization of the problem that *PriDIER* framework solves is as follows:

Problem: *Given a set of infected users trajectories T_i and non-infected users trajectories T, detect the close contacts between each non-infected trajectory and all infected ones, and measure the exposure risk from multiple infected sources, while preserving the privacy of the users.*

3.2.1 Contact Tracing

Contact tracing (CT) [17] is a category of analytics that determines whether two objects have come within close distance of (i.e., intersects with) one another for a certain period of time. In the context of the COVID-19 pandemic, the US Centers for Disease Control and Prevention (CDC) has broadly defined the following three criteria for two people to be declared in *close contact* indoors or outdoors: (i) they both were within 6 feet (≈1.8 meters) of one another without wearing masks; (ii) one of the two persons is infected, which is considered up to two days before the infected person has developed symptoms; and (iii) the cumulative total time of contact is 15 minutes or more [19].

Clearly, the first requirement is spatiotemporal, where the individuals have to be within the same space at the same time, while imposing a constraint on the spatial dimension (i.e., 6 feet). The second criterion captures the *exposure* to the virus, which is mandatory for the infection to take place. The third criterion is the duration of contact(s), which captures the duration of exposure to the virus in a *cumulative* fashion. The total durations of exposure is based on estimation of the viral load and a relative metric called "quanta", which can be translated to the probability estimates of viral infection[1] [14]. In other viral respiratory infections contexts, the CDC criteria for *close contacts* can be applied for the purpose of contact tracing with possible differences in the exact temporal duration and spatial distance requirements.

There are two forms of spatial intersection that must be considered when performing contact tracing: *(i) direct contact*, and *(ii) indirect contact.* Direct contact occurs when an infected and susceptible person share the same location at the same time, while indirect contact occurs when the two individuals share the same location at different times within a given interval. Based on the CDC criteria discussed above, there is a need to calculate an individual's total risk of viral exposure by considering the *cumulative* durations of all contacts both direct and/or indirect.

To demonstrate the challenges of contact tracing, that requires the implementation of an aggregate spatiotemporal join query, consider the following scenario shown in Figure 15 [36].

Example 3: *Consider three trajectories $\{A, B, C\}$ representing three different people walking in a building. Each consists of multiple points (i.e., $A = \{a_1, \ldots, a_{11}\}$, $B = \{b_1, \ldots, b_{11}\}$, $C = \{c_1, \ldots, c_{11}\}$). One person is infected with COVID-19 and the infected trajectory is B (i.e., the green dotted line), and A has 3-second stops (i.e., a point with 3 seconds duration), and C has 1-second stops.*

[1] CovidReduce: https://db.cs.pitt.edu/covidreduce

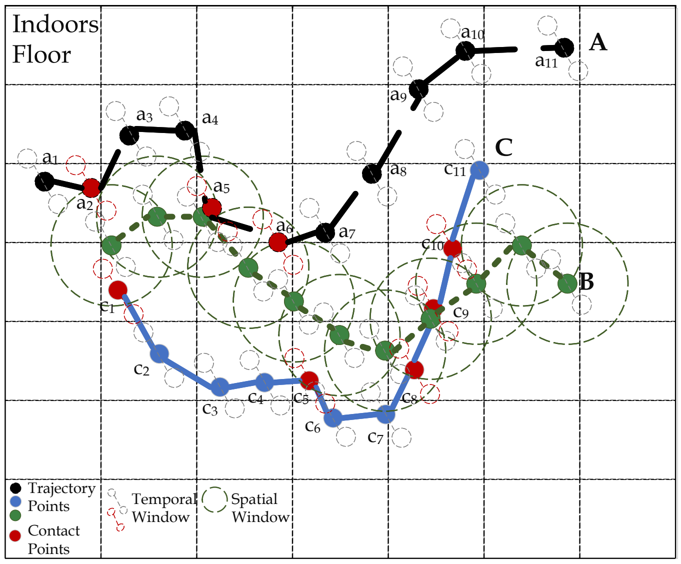

Figure 15: Example of direct and indirect contacts.

By visually examining the trajectories, we can easily identify the direct contacts, which are $\{a_5, c_9\}$ *and the indirect contacts, which are* $\{a_2, a_6, c_1, c_5, c_8, c_{10}\}$*, but not the total duration of all the contacts. The accumulated duration of the contacts define the degree of exposure, and according to the CDC guidelines, to find the contacts we need to consider a spatial window (i.e., the green dashed circles) and a temporal window denoted with the two non-filled points with a dashed outline and connected with a dashed line.*

Thus, A has the highest risk of exposure with accumulated duration of contact of 9 seconds, compared to C, which has 5 seconds.

In *PriDIER*, we consider both forms of contacts. The segmentation of the spatial layout into zones assures that indirect contacts will be considered. In addition, the employment of spatial enveloping (def. 5) and temporal enveloping (def. 4) in our framework will insure the consideration of indirect exposure to residual particles in the air after an infected user leaves a specific zone.

3.2.2 Threat Model

In a fully privacy protected system, under no circumstances any private information of an individual user is exposed to any other user and non-trusted actor. This information in *PriDIER* includes users' identities, and their current and past locations.

In the *PriDIER* framework, two actors interact in order to carry out the CT task, the Moving Objects (MOs) and the Trusted Central Authority (CA). Each actor stores and exchanges portion of the data needed for contact tracing.

- MOs store their own trajectory locally. This way the privacy of users is not jeopardized through moving their location data out of their device. The only case in which users' trajectories are moved out of their devices is in case of positive detection of infection. Then, MOs stream their own trajectories to the CA as required by law.
- The CA keeps a list of MOs anonymized identifiers in the system for authentication/identification and communication purposes. Also, CA stores locally the infected trajectories only that MOs voluntarily have streamed to it in an anonymized fashion. Lastly, The CA sends the anonymized identification of an individual user and their trajectory to another user only if the individual is positively infected.

In the context of CT, similar to an infected MO that sends to CA its trajectory, CA releasing the trajectories of infected MOs to non-infected MOs is not considered a privacy violation. Hence, neither MOs nor CA leak any private information.

3.2.3 Communication Protocols

In a mobile client-server environment, there are two ways data is transmitted between the server and mobile devices: *Push-based* and *Pull-based.*

3.2.3.1 Push-based Protocol

In an optimized Push-based protocol, the CA broadcasts the infected trajectories to all the MOs in the system. It is performed in three phases.

Phase 1: The CA sends out all the zones that the infected trajectories occupied (i.e., $\{Z_1, ..., Z_n\}$) to each participating MO in the system. Then each MO responds to the CA with the zones that overlap with the ones received from the CA in the first round.

Phase 2: For each MO, based on the received zones at the CA in Phase 1, the CA sends the timestamps along with all the zones Z_i that overlap with each particular MO's zones in the format

$$\{..., (Z_i, ts_i, c_i), ...\}$$

where c_i is the count of how many infected MOs have shared the same zone Z_i at the same timestamp ts_i overall.

Phase 3: Lastly, each MO conducts contact detection by executing a *temporal aggregation join* of its own trajectory and the received infected trajectory from the CA.

In case a close contact is detected, the MO sends back to the CA their own trajectory.

3.2.3.2 Pull-based Protocol

In an optimized Pull-based protocol MOs request the infected trajectories from the CA. It is performed in two phases.

Phase 1: Participating MOs send out a pull request message to the CA with the zones that they have occupied (i.e., $\{Z_1, ..., Z_n\}$). Then the CA sends response messages to MOs containing the timestamps along with all the zones Z_i that overlap with each MO zones sent in the pull request in the format $\{..., (Z_i, ts_i, c_i), ...\}$ where c_i is the count of how many infected MOs have shared the same zone Z_i at the same timestamp ts_i overall.

Phase 2: Each MO conducts contact detection by executing a *temporal aggregation join* of its own trajectory and the received infected trajectory from the CA. In case a close contact is detected, the MO sends back to the CA their own trajectory.

3.2.3.3 Discussion

There is a trade-off between an optimized communication between the CA and MOs and location privacy preservation. Specifically, MOs need to reveal their past locations to the CA for the the CA to filter out locations that a given individual (MO) has not visited. Following is an analysis of the optimized and non-optimized communication protocols in terms of *performance* and *privacy preservation* (see Table 4).

Performance: In terms of performance, the analysis focuses on: the number of rounds (exchanged messages), and the frequency of pulling (or pushing) trajectories.

The number of exchanged messages in the Push-based protocol is higher than its in the Pull-based. In the Push-based protocol, the number of messages is four compared to three messages in the Pull-based. This results in more processing and potential energy consumption at the MOs devices due to sending/receiving messages.

In the Push-based protocol, the frequency with which the CA sends (pushes) the infected trajectories to the MOs can be reduced to whenever new infected trajectories are available to be pushed. On the other hand, in the Pull-based protocol, MOs are oblivious to the availability of new infected trajectories. This results in potential pulling requests from MOs for infected trajectories even if there are no new ones to process.

Privacy Threat: Considering the threat model discussed in Section 3.2.2, when MOs share their locations (zones) without timestamps during the communication with CA, they do reveal their locations that they have occupied, even though they are not declared infected yet. This results in less costly processing at the MOs sides, since the amount of zones received from the CA will be reduced significantly to the only relevant ones.

The non-optimized pull-based and push-based protocols do not include the step where zones are sent by MOs to the CA. This results in each MO receiving the complete set of all infected trajectories from the CA for processing the *temporal aggregation join* which implies more trajectories to process, in addition to higher energy consumption due to longer transmissions of receiving longer longer messages from the CA.

Table 4: Push-based and Pull-based protocols usage trade-offs.

	Optimized Push-based	**Optimized Pull-based**	**Push-based**	**Pull-based**
Performance	- Four rounds - Only when new data is available - Relevant subset of infected trajectories	- Three rounds - Potential unnecessary requests - Relevant subset of infected trajectories	- Three rounds - Only when new data is available - Complete set of infected trajectories	- Two rounds - Potential unnecessary requests - Complete set of infected trajectories
Privacy Threat	Revealing zones	Revealing zones	None	None

3.2.4 MO components

The MO side of *PriDIER* consists of three components (Figure 16).

Query Processor In this component, the actual processing of the contact tracing operations takes place. There are multiple ways to execute the *temporal aggregation join* of the local trajectory and the received infected trajectory from the CA to determine close contact, each utilizing different optimizations and performance characteristics. The input is the infected trajectory one point at a time (① in Figure 16), and the final output is the input infected MO points at which the two MOs were in contact in a specific location at a certain time, and the total cumulative duration of contacts (⑤ and ⑥ respectively in Figure 16). The duration output, and the max timestamp are fed to the next component, namely the "Exposure Aggregator" (④ in Figure 16).

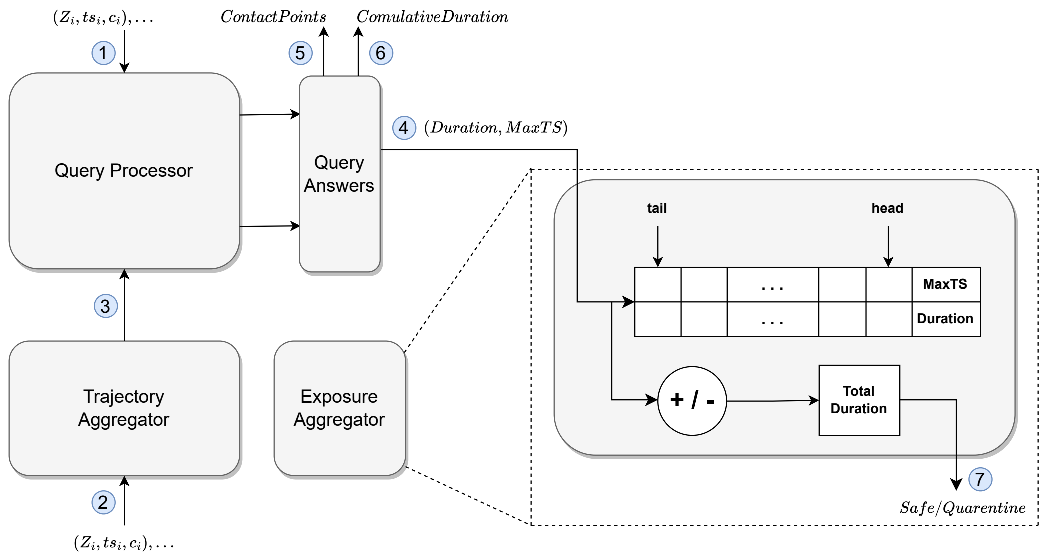

Figure 16: *PriDIER* at the MO side.

Exposure Aggregator The key observation underlying this component is that viral load is not attributed to a single close contact with a specific infected MO. A specific MO can be exposed to significant viral load through contacting many different MOs while not having any close contact with any of them. For example, having a contact of 10 seconds with 90 different infected people within two days can lead to high viral exposure. In the case of COVID-19, according to [14], $\sigma = 2$ (i.e., two days is the time window at which we keep accumulating the viral load).

The Exposure Aggregator keeps track of all the exposures in seconds for the past σ days and calculates the viral load across MOs cumulatively—not only cumulatively within a single infected MO, but cumulatively across multiple MOs.

The Exposure Aggregator can be implemented using a linked list with head and tail pointers (Figure 16) and a short integer variable. For each *temporal aggregation join* conducted with infected trajectories (output of the Query Processor), the aggregated duration along with the latest timestamp of contact are appended to the tail, and the short integer variable is incremented with that duration. Each time a node is appended at the tail (temporal aggregation and maximum timestamp), the head is checked. If the timestamp is more than two days earlier than the current time, the head node is removed and the node's duration is subtracted from the short integer that records the total duration of viral exposure across MOs. For each input to the component, it determines and outputs whether there is a risk of contracting the virus or it is safe by passing the total duration to the HealthDist Quanta Calculator [14] (⑦ in Figure 16).

Trajectory Aggregator This component provides an efficient buffering mechanism for the received infected trajectories that helps eliminating repeated processing of trajectory points while processing *aggregated temporal joins*. When the MO is busy processing points that belong to a certain infected MO trajectory, there is a possibility of having more infected trajectories arriving at the MO from the CA (② in Figure 16). This potentially occurs in case there are multiple CAs in different environments, where participating MOs send multiple Pull-based requests to CAs and then receive responses from multiple CAs. Another context at which this occurs is if the communication protocol employed is a Push-based where the CA broadcasts the infected trajectory on regular basis regardless of the number of CAs (Section 3.2.3). Typically, infected trajectories are processed sequentially in the order with which they were received. The issue with this approach is the potential processing of points multiple times. This could happen if the received trajectories belong to MOs that already were in contact. This is inevitable since the trajectories at risk that are sent to the CA are the ones that contacted other infected (or at risk) ones.

Thus, the Trajectory Aggregator combines and accumulates the arriving trajectories such that it feeds the Query Processor only unique points, shared by multiple trajectories (③ in Figure 16). In effect, it performs a union operation, which can be implemented using a simple unordered hash table that keeps counting the number of times a specific point (zone and timestamp) trajectory belongs to an infected trajectory. This way, each point in this hash table is processed (fed to the Query Processor) once, while accounting for the duration to be the total times the point has been received.

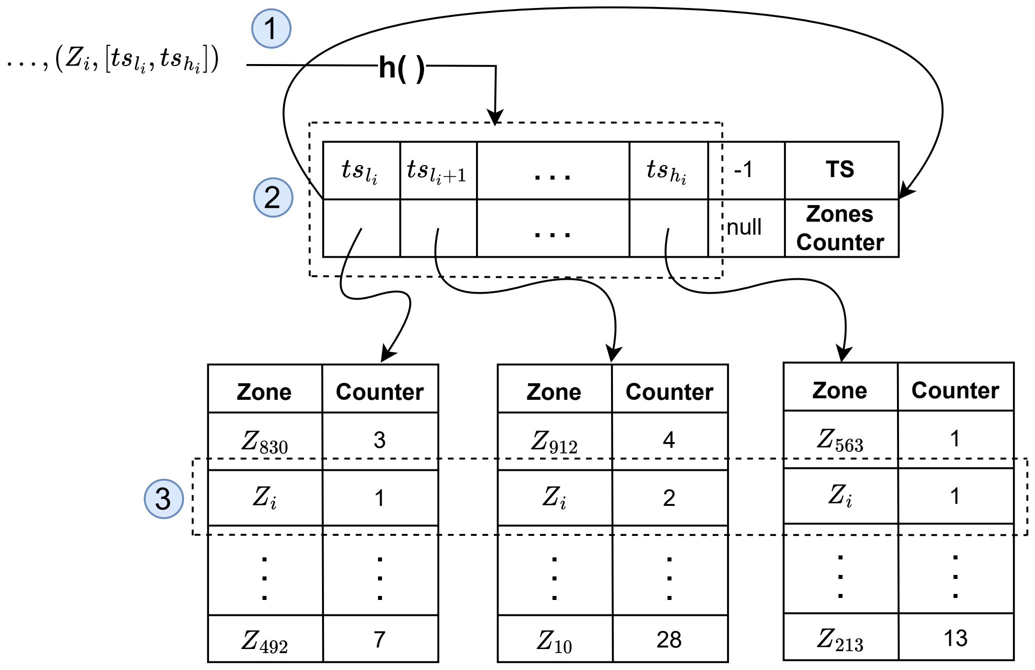

Figure 17: Global Infected Trajectory component in the CA.

3.2.5 CA Components

The CA side of *PriDIER* consists of two components.

Global Infected Trajectory Figure 17 shows this component that accumulates and combines all the infected trajectories, received from MOs, into a large single infected trajectory, namely *Global Infected Trajectory*. This global trajectory stores the information of all infected trajectories in the past W_t seconds, where W_t is the time window that is relevant for contact tracing. For each second in W_t, no known infected MOs ever existed in any zone in the system, multiple MOs existed in a single specific zone (e.g., a meeting room), or multiple MOs simultaneously existed in different zones at the same timestamp. This results in having for each timestamp (second) no zones, single zone, or multiple zones. To account for the exposure measure, we count the number of infected MOs that existed in the same zone at the same timestamp.

This component can be implemented using two circular buffers of fixed size arrays. The size of the two arrays is the number of seconds (timestamps) in the time window W_t. The first array stores the actual timestamps, and the second stores a pointer to a hash table that stores zones with counters. The zones are the ones at which MOs occupied in the system while they were infected (or at risk), and the counter is number of times the zone was received as part of a trajectory at that corresponding timestamp.

Each time an infected trajectory is received, the timestamp is hashed into an element in the circular buffer (① in Figure 17), the timestamp (TS) (② in Figure 17) and the hash table of that array element is updated for that specific zone. If an entry exists for that zone, then the counter is incremented. Otherwise, the zone is stored in the hash table with a counter value of 1 (③ in Figure 17).

Checkpoint Keeper This component prevents re-sending the same infected trajectory points to MOs. By keeping track of the latest sent timestamp for each MO, subsequent responses to MOs will not contain the parts of the Global Infected Trajectory that were already sent to a given MO.

Specifically, this component is a look up table that keeps the latest timestamp that each participating MO has received so far from the CA and can be implemented simply by having a single hash table. In a steady state, the structure is updated for a specific MO whenever the CA sends the Global Infected Trajectory to that MO. That entry is not affected by the arrival of new infected trajectories at CA as long as each timestamp in this structure is older than the oldest timestamp in the recently arrived trajectory. If the latter is not the case, the structure is rest by updating all the timestamps in all entries to match the oldest timestamp in the newly received infected trajectory. This ensures that subsequent responses to MOs with the Global Infected Trajectory include the newly received infected trajectory.

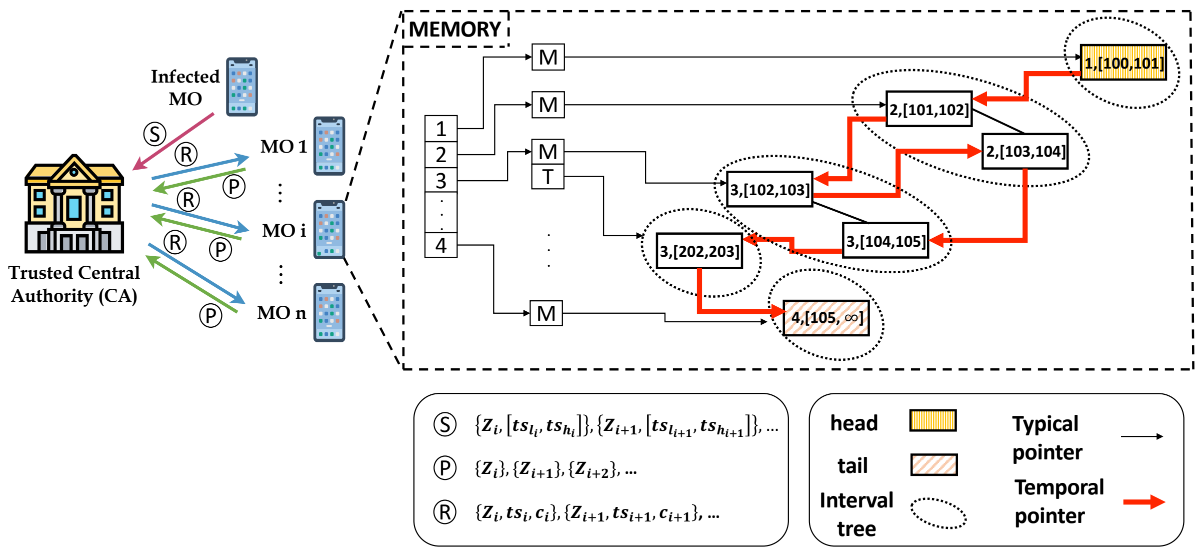
Infected MO
Trusted Central Authority (CA)
MO 1
MO i
MO n
MEMORY
1
2
3
4
M
T
1,[100,101]
2,[101,102]
2,[103,104]
3,[102,103]
3,[104,105]
3,[202,203]
4,[105, ∞]
S {Z_i, [ts_{l_i}, ts_{h_i}]}, {Z_{i+1}, [$ts_{l_{i+1}}$, $ts_{h_{i+1}}$]}, ...
P {Z_i}, {Z_{i+1}}, {Z_{i+2}}, ...
R {Z_i, ts_i, c_i}, {Z_{i+1}, ts_{i+1}, c_{i+1}}, ...
head
tail
Interval tree
Typical pointer
Temporal pointer

Figure 18: *e-Racoon* access structure for MO i that has occupied zones 1, 2, and 4 on a Monday (M) and zone 3 on Monday (M) and Tuesday (T). Note the data exchange protocol between MO and CA via the messages Ⓢ (Send), Ⓟ (Pull), and Ⓡ (Response).

3.3 Temporal Aggregation Join Query

The *temporal aggregation join* query captures the aspects of contacts (including close contacts) for respiratory diseases. That is, it answers whether two people were in contact, and how long they remained in contact over multiple instances (i.e., cumulatively). Formally:

Definition 8. *Temporal Aggregation Join (TAJ) Query: Given two trajectories T_l (size n) and T_q (size m), find the cumulative temporal durations of contacts between T_l and T_q within the previous W_t seconds window, considering a spatial window of ϵ meters.*

Recall, T_l is the MO local trajectory and T_q is the query trajectory sent by CA. In case the cumulative temporal duration of contact $= 0$, then no contact has occurred. In case the cumulative temporal duration of contact $\geq \delta$, then a close contact has occurred.

3.3.1 TAJ Query Processing

While executing TAJ query for CT and exposure risk, an MO needs to consider localization inaccuracies and indirect contacts due to residual air particles (or exposed surfaces) after an infected person leaves a zone. We employ spatial and temporal enveloping (Def. 5 and Def. 4) to account for both. Figure 12 shows an example of the effect of spatial enveloping. Even though MO_2 does not cross path directly with $MO_{infected}$, they cross path at the enveloped space of $MO_{infected}$ (red and grey zones), which we consider as a contact.

A TAJ query could be processed by different approaches. The naïve approach is to store the local trajectory in a raw format, and use a nested loop join on both spatial and temporal dimensions (i.e., zone and timestamp) followed by aggregation (i.e., sum).

An optimized approach utilizes in-memory indexing to store the local trajectory to speed-up the join operation. One way is to employ a typical spatial index (e.g., Kd-tree), and then process temporal data as is with no optimizations. Another alternative is using a temporal index to index timestamps (e.g., Interval trees) and then process the spatial data as is. Clearly, using a temporal index will be less efficient since the data is temporally dense. That is, a MO will always occupy a specific place at each second, but it will not exist at more than one place at the same time (timestamp). However, a MO could occupy the same zone at different times.

Both approaches optimize the join in one dimension and require a further processing step in the other dimension. This observation led us to develop *e-Racoon*, an access structure that optimizes both dimensions and is tailored to TAJ query processing.

3.3.2 The e-Racoon Access Structure

In this section we present our novel access structure, dubbed *e-Racoon*, that efficiently processes the TAJ query (Def. 8), discussed briefly in [36], and inspired by the Racoon access method [2].

Overview *e-Racoon* is a multi-level in-memory access structure (Figure 18) that stores zones and intervals. The first level is a spatial grid that stores zones that MOs occupy. Each spatial bucket in the grid stores a second temporal level. The granularity of this temporal level (e.g., day, half day, or hours) varies based on the application/data. In our case of CT, we pick days. The temporal bucket stores a single pointer to a data structure that stores time intervals during which MOs occupy a particular zone on a particular day (the spatial and temporal buckets). We choose *Interval trees* [39] as an augmented data structure to store intervals, and *Red-black trees* [11] as the underlying implementation because of their self-balancing feature that keeps the cost of their traversal predictable.

Each node of an Interval tree stores a zone, a time interval, and three pointers. The intervals are needed for answering exposure measuring queries and two pointers are used for the left and right children nodes of the Red-black tree. The third pointer points to the node which contains the interval that temporally proceeds the current (stored) one (thick, red arrows in Figure 18). As discussed in Section 3.3.3, these *temporal pointers* could potentially point to a node that exists in a different tree, since a MO has to change their location to mark the time of departure of a certain zone.

e-Racoon stores *intervals* in the form: $[t_{arr}, t_{dep}]$, where t_{arr} is time of arrival and t_{dep} is the timestamps of departure from a particular zone. As location is being sampled by a MO, points sampled consecutively at the same zone are not inserted in the structure, and the MO is considered in the same location until the MO leaves that zone (i.e., the sampled location is of a different zone). This reduces the memory footprint, and alleviate the computational processing at the MO side.

e-Racoon keeps a global *tail* pointer that points at the very last node stored in the whole structure (latest interval in the local trajectory). The most recent node will have an interval that has its t_{dep} marked as '∞', which implies that the user is still at the same zone. This happens when a subsequent node is ready to be inserted (i.e., MO left the current zone).

Insertion When a MO changes zones, a new Interval tree node is inserted with a new zone, and mark the beginning of the interval to be the time of arriving at the zone (i.e., t_{arr}) and the end of the interval to be ∞. After that, we look up the *tail* of *e-Racoon* (most recently inserted node in the structure) and update the end of the interval from being '∞' to $t_{arr} - 1$ and update the *temporal pointer* to point at the newly created node.

Inserting the node is done by looking up the bucket of the new zone, then retrieving the temporal bucket that represents the day at which we insert the node. In case no bucket exists that resembles the day, a new one is created, a new empty Interval tree is initialized, then a typical Interval tree insertion is performed.

Deletion and Update *e-Racoon* does not update or delete nodes in Interval trees, because of the nature of trajectory data (i.e., no change of time or location in the past). Instead, we expire (i.e., delete) a whole tree that is stored in the bucket that resembles a day. Data is kept in the *e-Racoon* structures for certain number of days depending on the contact tracing time window. For instance, in case of COVID-19, we need only to keep the data for a week. For safety, an extra week could be stored. Deletion is performed by traversing the tree and freeing the memory back to the system. Thus, no delete or update re-balancing.

Search The search operation implements the join of two trajectories and maps looking up if a point in the infected trajectory results in contact with the local trajectory to the stabbing problem [45]. First, the zone of the received point is looked up. If the spatial bucket for that zone is not found, it is declared a no contact. Otherwise, the temporal bucket is looked up from the point's timestamp. If it exists, the timestamp is searched in the Interval tree via stabbing. In case the timestamp stabs an interval, a contact is declared, and the duration is computed as part of the result of the query.

3.3.3 Optimizations

We discuss two *e-Racoon* and one *PriDIER* optimizations.

***e-Racoon* Temporal Pointers** Temporal pointers to support temporal aggregation (i.e., sum of intervals) to compute durations of CT are inspired by three observations:

1) Spatial and temporal locality aspects of human trajectory contacts. If two MOs are in contact at a certain time and location, it is highly likely that contact will last for some period of time. To reduce the access to the structure through the spatial and temporal buckets layers, and stabbing for each point, we access the temporal pointer in the last node (with stabbed interval) and compare the next points in the received trajectory linearly.

2) String searching. Computing the duration of the join of two trajectories resembles the longest common substring in string searching. This will result in processing the *temporal aggregation join* query similar to comparing two trajectories using string searching. However, the structure is accessed again (stabbing Interval trees) whenever a mismatch is encountered in the linear search, providing a means to efficiently jump to the next point of contact.

Lastly, *3) Range query support in the B^+-tree.* Linking the nodes of the access structure across trees and buckets is inspired by the B^+-tree leave nodes level linked list. In the B^+-tree, the leave nodes level is linked forming a linked list that sorts the keys in the leave nodes. In *e-Racoon*, the order of linking is temporal.

Therefore, linking the nodes in *e-Racoon* results in turning the time complexity of processing the query from linearithmic time (accessing the structure and stabbing for all received points) to logarithmic and linear (accessing the structure through the two layers once and then linearly traversing the rest of the nodes) assuming that both trajectories are in full contact. The full complexity analysis is in Section 3.3.4.

Moreover, having the global 'head' pointer to the very first node inserted and a global 'tail' to the very last node inserted, facilitates efficient linear traversal of the structure in temporal order to reconstruct and stream the local MO trajectory to the CA (message Ⓢ in Figure 18). This is achieved by accessing the global 'head' node, linearly and temporally ordered traversing of the nodes, and terminating at the global 'tail' node. The trade-off is that *e-Racoon* requires the Query Processor component in the MO side of *PriDIER* (Section 3.2.4) to have the input trajectory points temporally ordered.

***e-Racoon* CT Data Exchange** *e-Racoon* optimizes the contact tracing protocol discussed in Section 3.2.3, since the MO local trajectory will be stored in the format:

$$\{..., (Z_i, [ts_{l_i}, ts_{h_i}]), ...\}$$

This results in a reduction of points sent by using intervals instead of timestamps. An example with the messages Ⓟ, and Ⓡ, and Ⓢ is shown in Figure 18.

CT Early Termination Inspired by [41, 3], we reduce the cost of processing the *temporal aggregation join* query by stopping the processing once reaching a duration that meets the

Table 5: *e-Racoon* operations' time complexity.

***e-Racoon* Operation**	**Time Complexity**
Point Searching	$\mathcal{O}(\log(m))$
Contact Searching	$\mathcal{O}(\log(m)(n-\beta+\gamma)+\beta-\gamma)$
Point Insertion	$\mathcal{O}(\log(m))$
Tree (size m) Deletion	$\mathcal{O}(m)$

contact tracing criteria (i.e., $\geq \delta$). For example, in the case of COVID-19, reaching the cumulative contact duration of 15 minutes (900 seconds) across all contact incidents fulfills the close contact criteria. Thus, the query processing is terminated, and the local MO trajectory is sent to the CA for further contact tracing querying.

3.3.4 Complexity Analysis

Here we discuss the time complexity of the *e-Racoon* structure. The summary of the time complexity analysis is shown in Table 5.

Point Searching Since humans are not able to occupy different locations at the same timestamp (second) while walking indoors, in *e-Racoon* there will be at max a single interval that a point potentially stabs in the structure. This means that the complexity of finding whether a point results in a contact involves four steps.

First, the spatial hashing bucket look up is of $\mathcal{O}(1)$. Second, in case the spatial bucket exists, the temporal bucket within the spatial bucket look up is of $\mathcal{O}(1)$. Third, in case the temporal bucket exists, it will contain a pointer to the root of the target Interval tree to perform the stab operation. This is a search in a Red-black tree, which is known to have a complexity of $\mathcal{O}(\log(m))$ [11], where m is the number of nodes in the Red-Black tree. Since each Interval tree in the *e-Racoon* structure stores the intervals of a specific day, m is the number of intervals stored in the Interval tree for a specific zone during a specific day. Thus, m can be different for each zone on different days. For simplicity, we assume all Interval trees in the structure to have m nodes. This results in an *e-Racoon* search complexity of $\mathcal{O}(1 + 1 + \log(m))$. We simplify this to be $\mathcal{O}(\log(m))$.

Other approaches have different complexities. Starting with the spatial indexing typically using a Kd-tree, the tree will contain a single node for each unique spatial zone in the trajectory. That is, if we have z number of unique zones, then the tree will contain z nodes, and each node will store linearly all the timestamps at which the zones were occupied. This results in a complexity of $\mathcal{O}(\log(z) + t)$, where t is the number of timestamps in each node. It is clear that the value of t will be different for each node, but for simplicity, we assume that t is the same, and we notice that t is the dominant term, which makes point searching using a Kd-tree of a linear complexity.

In case of the typical Interval tree indexing, the tree will have a node for each point in the trajectory. That means a node for each unique timestamp. This results in having a large and dense tree to store. Thus, in case of a trajectory of size n, the time complexity to search for a point will be $\mathcal{O}(\log(n))$.

Contact Searching Without loss of generality, we provide the *e-Racoon* time complexity analysis by defining some variables. In a pair of trajectories, we have α segments that form continuous points of contacts. The total number of contact points across segments is β points. Out of the β points, γ points will be stabbed. Those are the first points in each continuous contact segment (See Figure 19. Note that γ will always be equal to α, since each continuous contact segment will contain exactly a single point to be stabbed (the very first point of the segment).

For example, having a pair that has all its points as a contact means the whole pair is a single segment of continuous contacts that has $\alpha = \gamma = 1$ and $\beta = n$, where n is the total number of points in the pair (i.e., length of each trajectory), and having a pair that has no contact at all means having $\alpha = \gamma = \beta = 0$. Note that a single point of contact in a pair is considered a continuous contact segment on itself, and it will be part of the β total points, part of the α number of continuous contact segments, and part of the γ points to be stabbed for that pair.

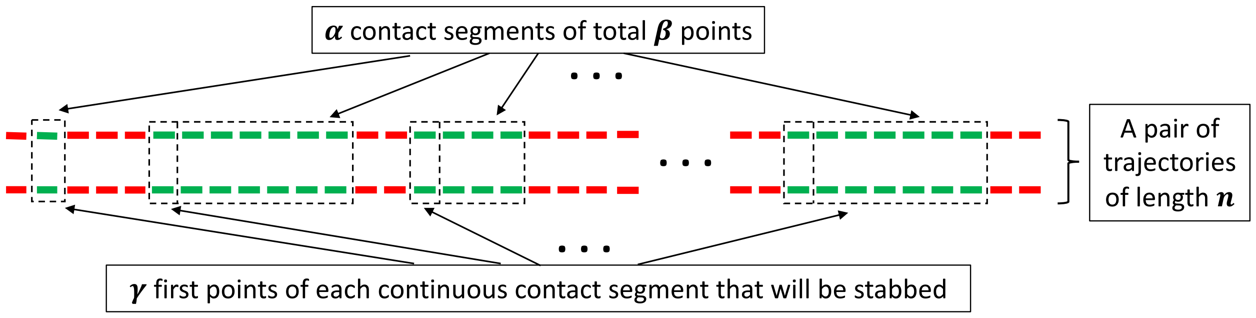
α contact segments of total β points
A pair of trajectories of length n
γ first points of each continuous contact segment that will be stabbed

Figure 19: Illustration of a pair of trajectories of length n, with α contact segments, γ points to be stabbed, and a total of β contact points across segments.

For the non-contact points, it is clear that each one of them $(n-\beta)$ will be stabbed. This is of a complexity $\mathcal{O}((n-\beta)\log(m))$. For the contact points (β), the first point at each segment (γ) will be stabbed, and then the rest of the points in each continuous contact segment is processed linearly $(\beta-\gamma)$ using the temporal pointer. This results in $\mathcal{O}(\gamma\log(m)+\beta-\gamma)$, where the $\gamma\log(m)$ is a result of stabbing the first point in the α continuous contact segments, and $\beta-\gamma$ is the remaining points in the continuous contact that is processed linearly. Therefore, we have the complexity $\mathcal{O}(\gamma\log(m)+\beta-\gamma+(n-\beta)\log(m))$. The result in a more summarized form is

$$\mathcal{O}(\log(m)(n-\beta+\gamma)+\beta-\gamma) \tag{4}$$

The best-case scenario is when all points are in contact. This means that $\alpha=1$, $\gamma=1$, and $\beta=n$. That means the whole pair is a single continuous contact segment, there is only a single point to stab, and all points are contact points. Substituting the numbers for Equation 4 results in a complexity of $\mathcal{O}(\log(m)+(n-1))$ or $\mathcal{O}(n+\log(m))$, which reflects stabbing the very first point in the trajectory, and then linearly processing the remaining points. On the other hand, the worst-case scenario is when all points are a non-contact. That is $\alpha=\gamma=\beta=0$. Substituting the numbers for the same equation results in a complexity of $\mathcal{O}(n\log(m))$, which reflects stabbing all the points in the trajectory.

In case of a typical Kd-tree, recall that z is the number of unique zones in the trajectory, and t is the number of timestamps in each node (we assume it is the same for simplicity). This results in a linearithmic complexity of $\mathcal{O}(n(\log(z) + t))$ regardless of the contact information.

Similarly, In case of the typical Interval tree indexing, the time complexity to search for contacts will be $\mathcal{O}(n \log(n))$, which is linearithmic.

Point Insertion Inserting a point in *e-Racoon*, implies that the MO has changed zones. This is done in two main steps. First, the interval that belongs to the recently changed zone (the global 'tail') is updated by replacing ∞ with the new point timestamp - 1. This is the time of leaving the previous occupied zone. This is of a constant complexity, since a pointer to that global tail is kept. Second, a search of the new point is performed on the *e-Racoon* structure as discussed above. However, instead of searching the Interval tree, an insertion is performed. Since Interval trees are based on Red-black trees, the insertion complexity is $\mathcal{O}(\log(m))$ [11], so we have a point insertion complexity of $\mathcal{O}(\log(m))$.

In case of a typical Kd-tree, inserting a new point is of a complexity $\mathcal{O}(\log(z))$ regardless of the number of time stamps in the node, since the insertion will be always at the tail.

Similarly, In case of the typical Interval tree indexing, the time complexity to insert a point is $\mathcal{O}(\log(n))$.

Therefore, inserting n points into the *e-Racoon* structure, Kd-tree, and Interval tree results in an overall worst-case insertion complexity of $\mathcal{O}(n \log(m))$, $\mathcal{O}(n \log(z))$, and $\mathcal{O}(n \log(n))$, respectively.

Deletion and Update As discussed in Section 3.3.2, there is no deletion or update of nodes in *e-Racoon*'s tree structures. A whole tree is deleted when it expires (i.e., outside W_t) by traversing it linearly and free the dynamically allocated memory. Thus, we iterate through the spatial and temporal buckets, and for each Interval tree we free the memory allocated for each node. Thus, the deletion complexity is $\mathcal{O}(m)$.

On the other hand, In case of typical spatial or temporal indexing (Kd-tree or Interval tree), the deletion operation is costly. This is due to the lack of a handle to the nodes desired to be deleted. Therefore, the complexity is $\mathcal{O}(zt)$ and $\mathcal{O}(n \log(n))$ for the Kd-tree and Interval tree respectively. Recall that z is the number of unique zones in the trajectory,

t is the number of timestamps in each node in the Kd-tree, and n is the size of the trajectory. It is clear that *e-Racoon* is more efficient in deleting only the nodes needs to be deleted since $m \ll n$ and $m \ll z$.

3.4 Evaluation

In this section, we present our testbed and discuss the experiments that compare the different approaches in realizing the *PriDIER* framework and evaluating *e-Racoon*'s effectiveness.

3.4.1 Testbed

Infrastructure We implemented the MO's Query Processor that executes the TAJ query using the different approaches in our evaluation. All the approaches were coded in C++11. We ran the experiments on a machine with Intel i7 CPU with 8 virtual cores, running at 4 GHz, and 32 GB of RAM memory. The operating system used is macOS 11.7, and the compiler is Apple Clang version 12.0.0.

Datasets We use both synthetic and real-world datasets in our experiments.

Synthetic Dataset: We crafted our own synthetic datasets following the same format used in [13, 12]. We synthesize the datasets to show how different approaches perform under different real-world characteristics. Those characteristics are based on: (i) the length of the trajectories (number of points); (ii) the number of unique zones in a trajectory; and (iii) the duration of contacts in pairs of trajectories (i.e. α and β). The datasets are a total of 18 that are split into full contact and no contact datasets. For each split, we vary the number of unique zones (single zone, mixed, and all unique zones). We also vary the trajectory sizes (half working day, one working day, and two working days) to mimic a realistic 8am to 5pm working day hours. The sampling rate for the synthetic datasets is a single point per second. That is 14,400 points for a half working day, 32,400 for a full working day, and 64,800 for two working days. The detailed characteristics of all synthetic datasets are presented in Table 6.

Table 6: Characteristics of the synthetic datasets.

Dataset	**Trajectory Size**	**# of Unique Zones**	$\boldsymbol{\alpha}$	$\boldsymbol{\beta}$
DS_1	14,400	14,400	1	14,400
DS_2	32,400	32,400	1	32,400
DS_3	64,800	64,800	1	64,800
DS_7	14,400	3,601	1	14,400
DS_8	32,400	10,802	1	32,400
DS_9	64,800	21,604	1	64,800
DS_13	14,400	1	1	14,400
DS_14	32,400	1	1	32,400
DS_15	64,800	1	1	64,800
DS_4	14,400	14,400	0	0
DS_5	32,400	32,4000	0	0
DS_6	64,800	64,800	0	0
DS_10	14,400	3,601	0	0
DS_11	32,400	10,802	0	0
DS_12	64,800	21,604	0	0
DS_16	14,400	1	0	0
DS_17	32,400	1	0	0
DS_18	64,800	1	0	0

Real-world Dataset: We experiment with [53, 54], which is a collection of taxi trajectories in the city of Beijing. Ee segment the city of Beijing into zones of $2Km^2$ and we regard taxi trajectories as indoor human trajectories. We pick two pairs of trajectories of the lengths 82 points and 4004 points, and we name them BJ_S (for short) and BJ_L (for large), respectively.

Approaches We compare our proposed approach against other baseline approaches to implement TAJ queries, as well as the state-of-the-art approach. We discuss those approaches briefly next.

Naïve (NLJ): In this approach, trajectories are stored in raw format as temporally ordered points. Each point contains the zone and the timestamp at which the point was sampled by the localization device. We process the TAJ using a Nested Loop Join approach, where we compare each point from the received trajectory with the points in the raw local trajectory.

Temporal Index: We implement a single Interval tree that stores points as intervals. Note that there is no repetition of locations per second, since humans are not able to exist in more than one location at a certain second. We use the stabbing approach [45] of each point in the received trajectory to check if a contact exists.

Spatial Index: We implement a single typical Kd-tree that stores spatial points in nodes, and then stores timestamps at which MOs occupied that space linearly in a list. Contrary to time, humans are able to occupy the same space at different times. We use the zone's 2D dimensions as the Kd-tree dimensions in our implementation.

CHLZZ: This is the current state-of-the-art approach presented in [8]. We implement their approach in our framework, which is a spatial grid with a single Interval tree in each spatial bucket.

e-Racoon: We experiment with three variations of the *e-Racoon* structure. The original *e-Racoon* discussed in Section 3.3.2, and two variations with no temporal pointers, namely *Racoon BST* and *Racoon RBT*. Where BST means that the underlying implementation is a basic Binary Search tree, and RBT means that the underlying implementation is a Red-black tree.

Metrics We measure latency (response time) and storage (memory footprint) as metrics to evaluate our proposed approach.

Latency: We measure the wall-clock time consumed to fully process a TAJ query by an approach in milliseconds, and we measure the wall-clock time consumed to insert points into the access method of an approach in nanoseconds.

Memory footprint: We measure in bytes the amount of memory consumed by each approach during its life-cycle.

3.4.2 Experiments

We carried out six experiments using both the synthetic dsataset (DS_1 - DS_18) and the Beijing dataset (BJ_S and BJ_L). All the results are averaged from 10 runs, executing the TAJ queries serially. Note that the Naïve (NLJ) approach has no memory footprint and no point insertion latency since it has no in-memory data structures.

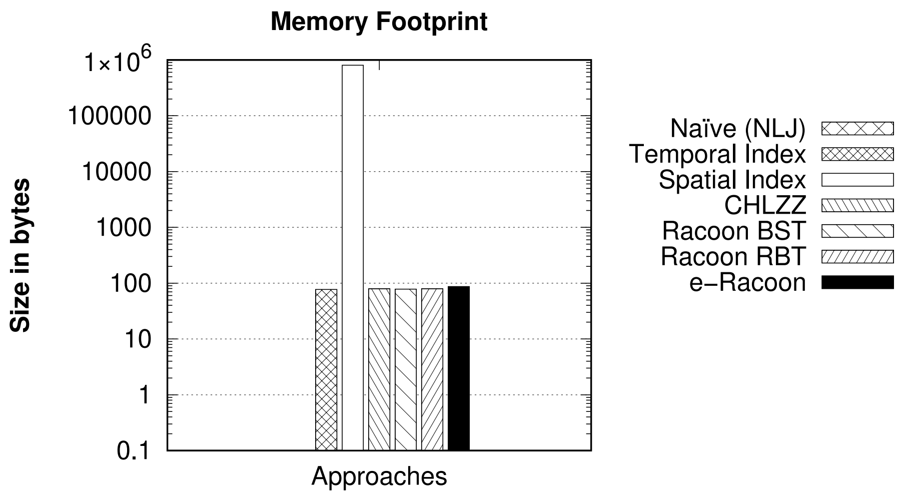

Figure 20: Memory footprint in bytes for datasets DS_13.

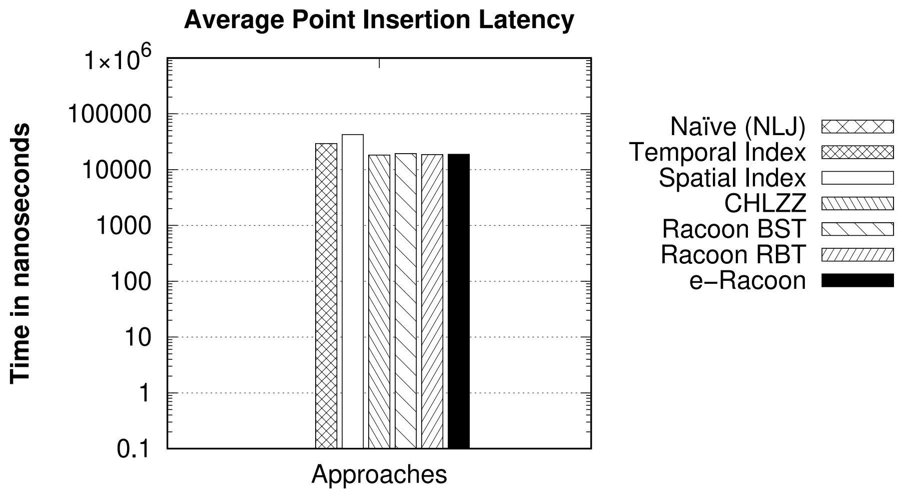

Figure 21: Average point insertion latency for datasets DS_13.

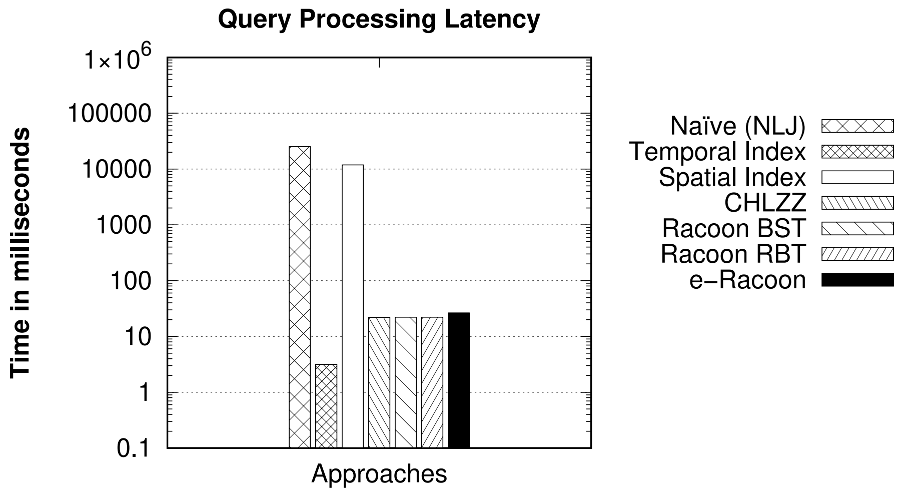

Figure 22: Query processing latency for datasets DS_13.

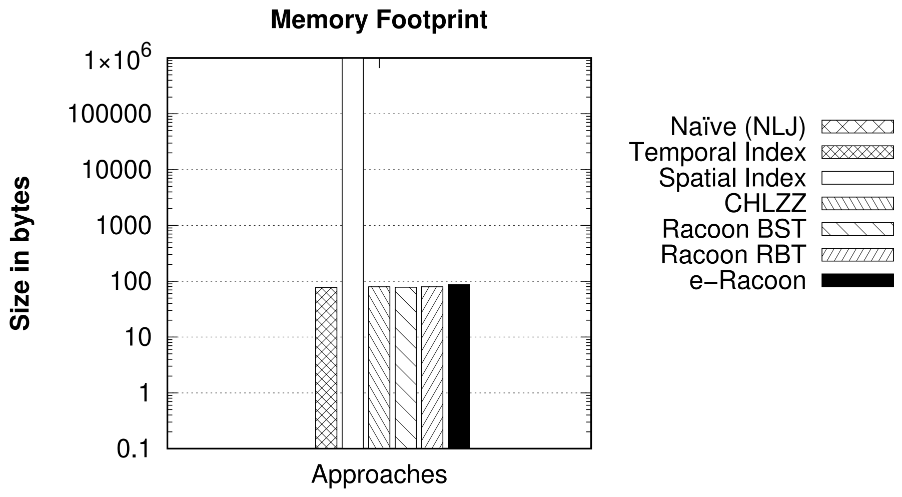

Figure 23: Memory footprint in bytes for datasets DS_14.

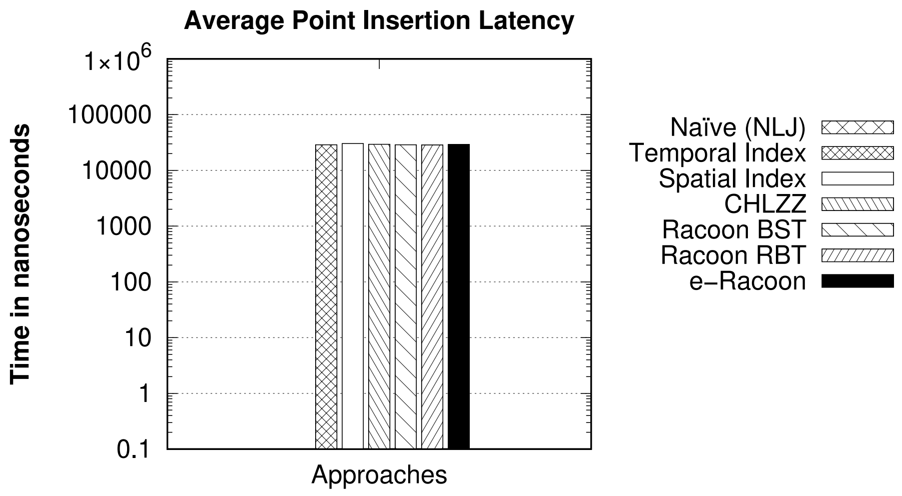

Figure 24: Average point insertion latency for datasets DS_14.

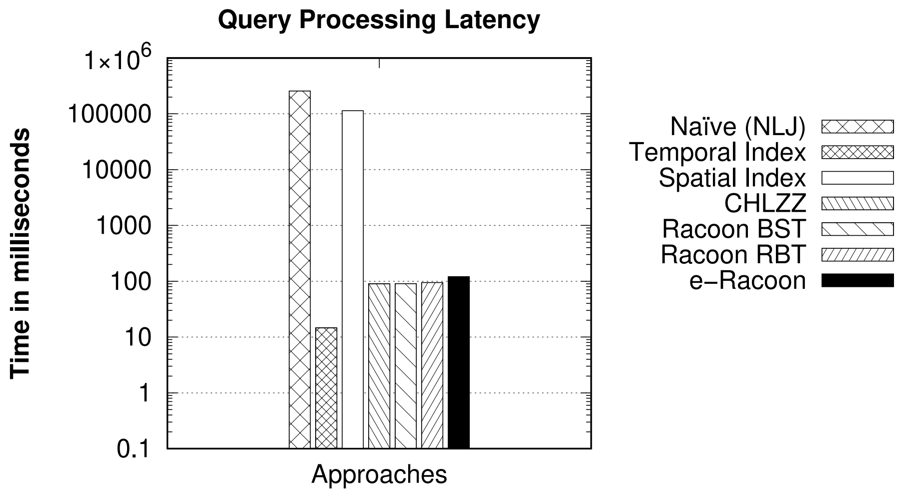

Figure 25: Query processing latency for datasets DS_14.

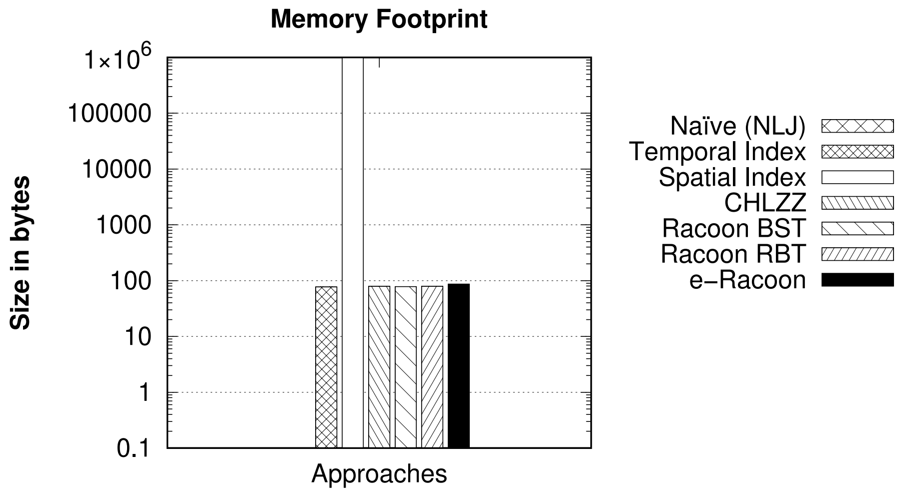

Figure 26: Memory footprint in bytes for datasets DS_15.

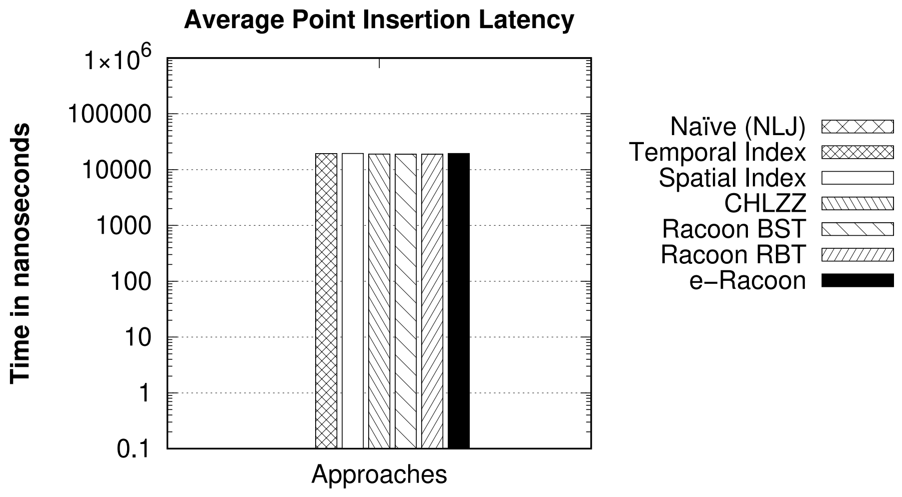

Figure 27: Average point insertion latency for datasets DS_15.

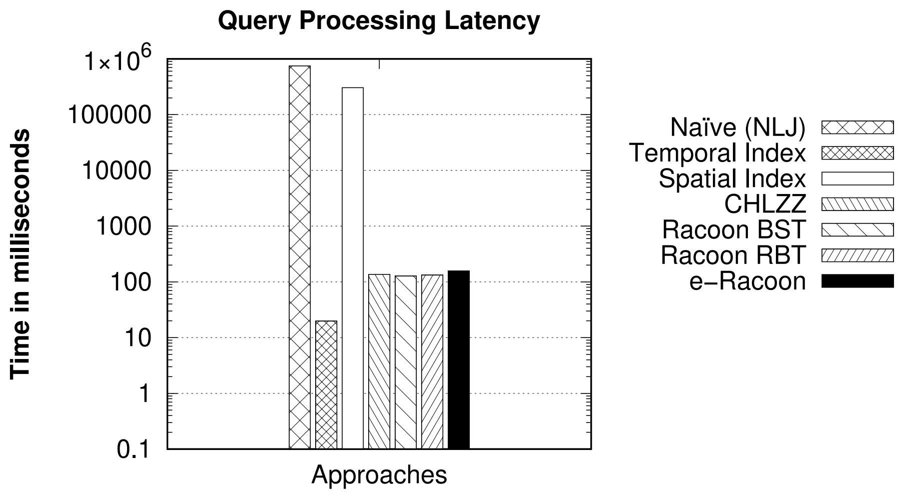

Figure 28: Query processing latency for datasets DS_15.

3.4.2.1 Experiment 1 (DS_13, DS_14, DS_15)

DS_13 exhibit an extreme case where MOs are fixated at a single zone. Note that the memory footprint is comparable for all approaches since all tree structures stores a single node of a single zone that is occupied for four hours (single interval), except the Spatial Index, where timestamps stored linearly (no intervals). CHLZZ has a single spatial bucket with a single node Interval tree. The same for the *e-Racoon* and its variations, but with an extra temporal layer of buckets. This results in Spatial Index processing latency close to the Naïve (NLJ) approach due to linear scanning of timestamps. The Temporal Index's processing time outperforms all the other approaches (3.15ms), since there are no hashing buckets and extra layers as in CHLZZ, and *e-Racoon* and its variations. *e-Racoon* performed slightly under its variations and CHLZZ (CHLZZ 22ms, Racoon RBT 22.12ms, and *e-Racoon* 26.39ms), due to the extra check for the temporal pointer for each stabbed point in hope for a contact occurrence. The point insertion latency is similar for all approaches. DS_14 and DS_15 exhibit similar results.

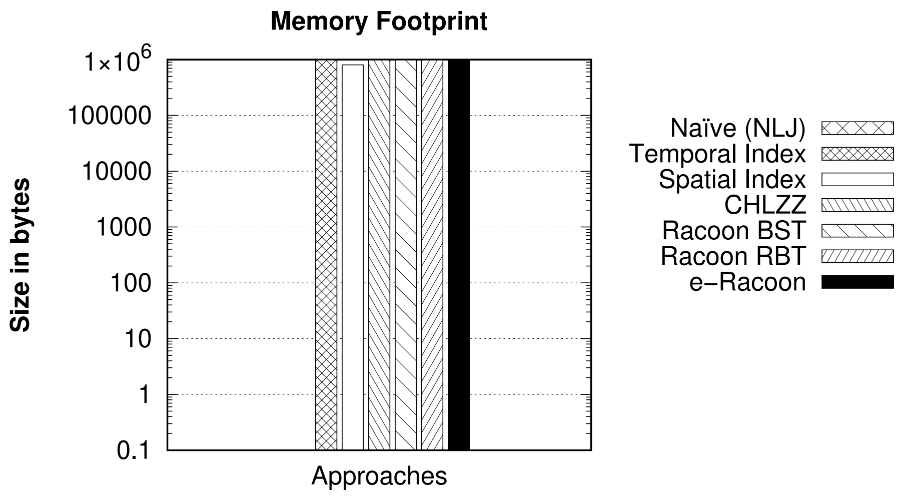

Figure 29: Memory footprint in bytes for datasets DS_1.

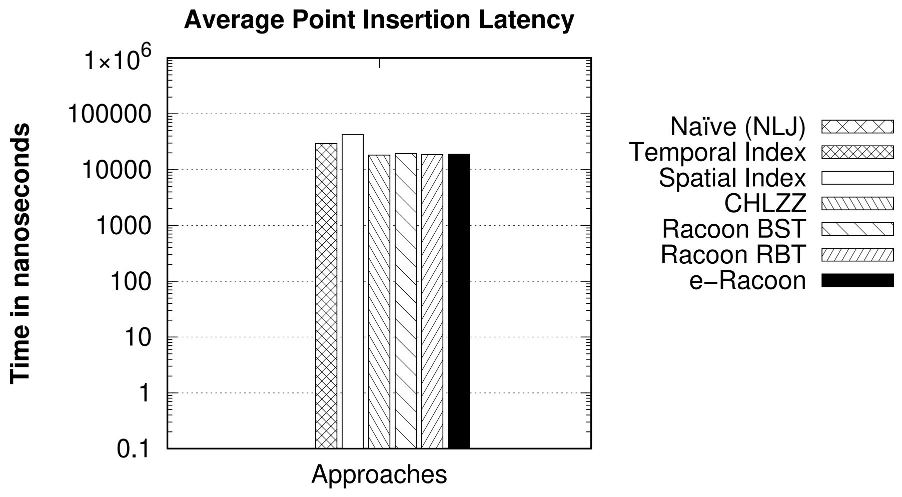

Figure 30: Average point insertion latency for datasets DS_1.

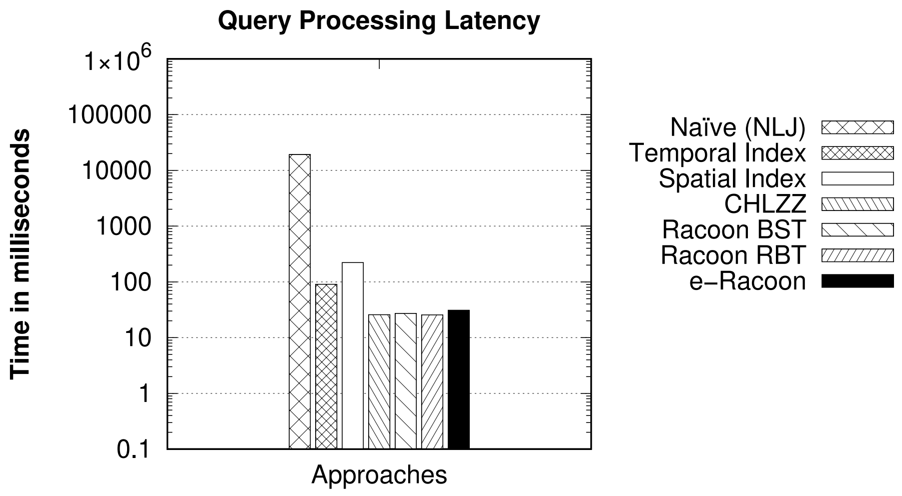

Figure 31: Query processing latency for datasets DS_1.

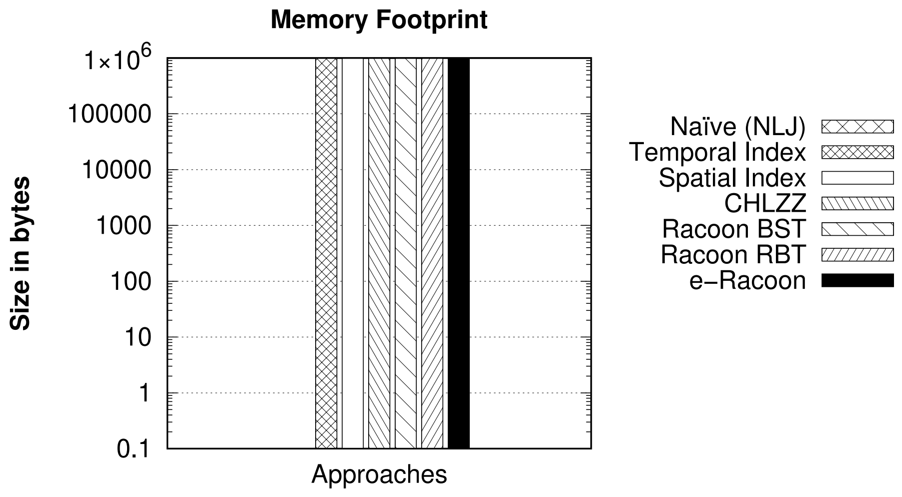

Figure 32: Memory footprint in bytes for datasets DS_2.

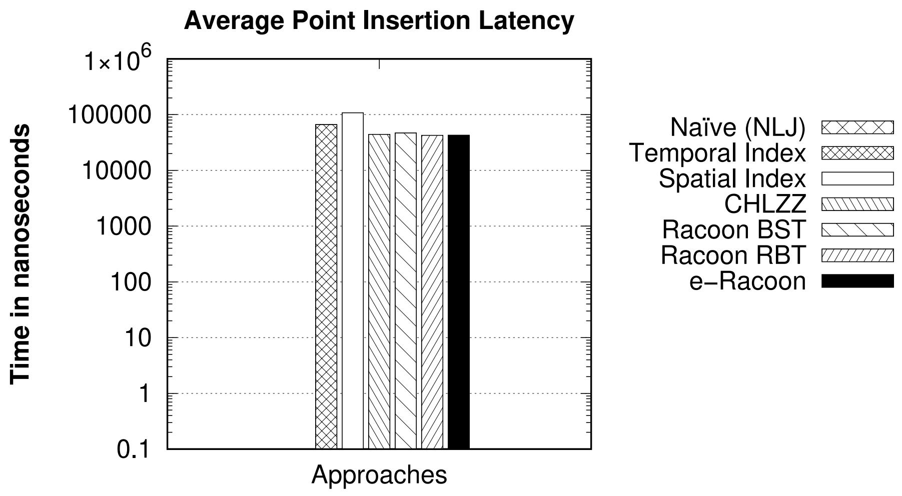

Figure 33: Average point insertion latency for datasets DS_2.

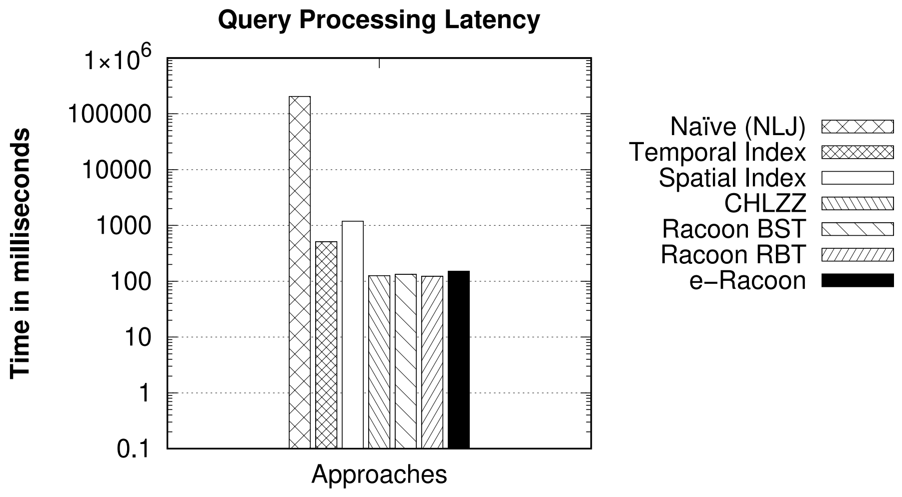

Figure 34: Query processing latency for datasets DS_2.

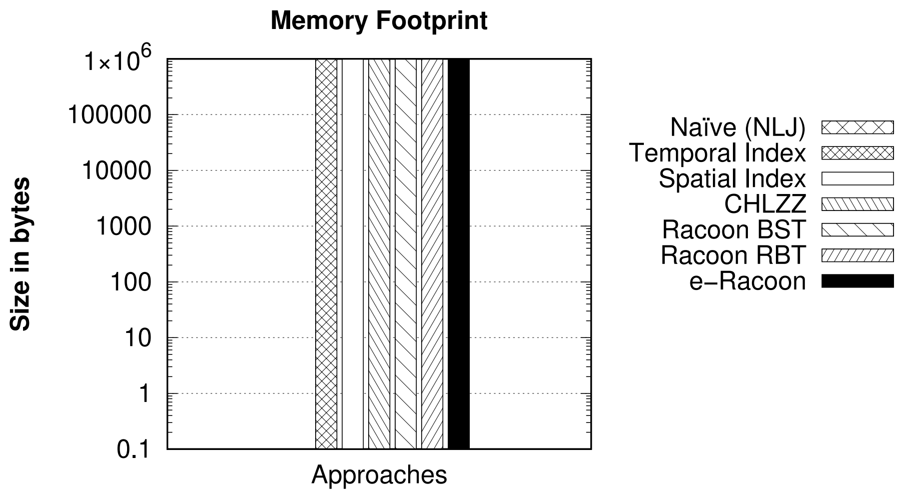

Figure 35: Memory footprint in bytes for datasets DS_3.

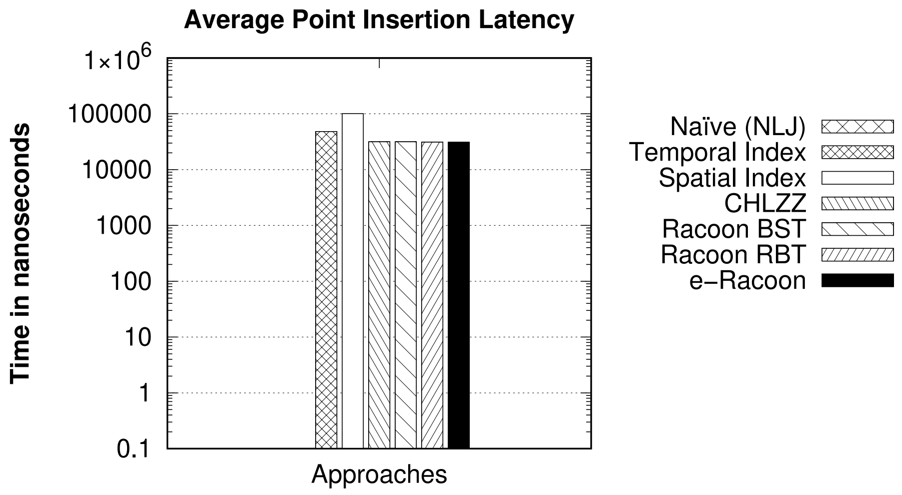

Figure 36: Average point insertion latency for datasets DS_3.

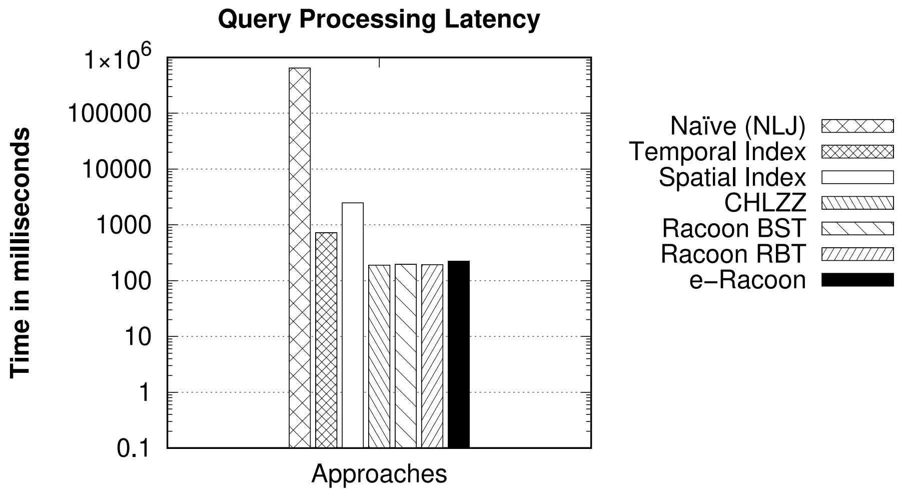

Figure 37: Query processing latency for datasets DS_3.

3.4.2.2 Experiment 2 (DS_1, DS_2 and DS_3)

DS_1 dataset is another extreme where MOs are constantly walking and occupying unique zones. The Spatial and Temporal Indexes consumed higher memory than the other approaches which have comparable results. The Spatial Index stores a single node for each point (14,400 nodes), each containing a single timestamp, and the Temporal Index stores a single node for each point (one second interval). The other approaches have 14,400 spatial buckets (with an extra temporal layer in *e-Racoon* and its variations). Each bucket will be associated with a tree of a single node (one second interval). This results in comparable memory footprint consumption across all approaches with the Spatial Index having slightly less consumption due to storing a single timestamp instead of an interval in nodes. The highest latency (processing and insertion) is by the Spatial Index and then Temporal Index respectively due to the larger tree sizes. CHLZZ, *e-Racoon*, and its variations have similar results to the Experiment 3.4.2.1. DS_2 and DS_3 exhibit similar results.

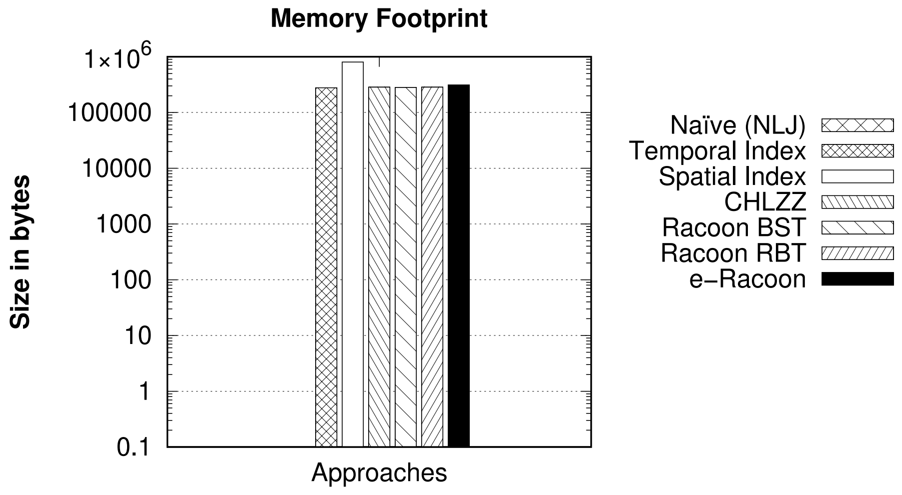

Figure 38: Memory footprint in bytes for datasets DS_7.

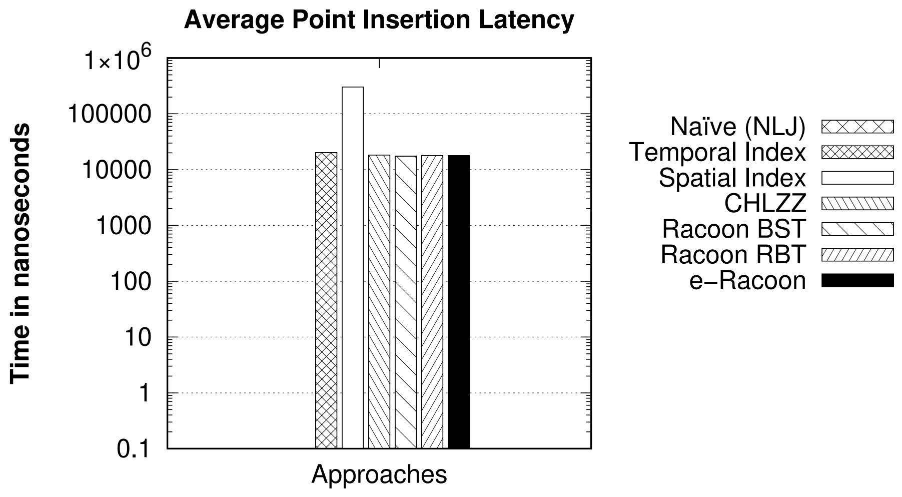

Figure 39: Average point insertion latency for datasets DS_7.

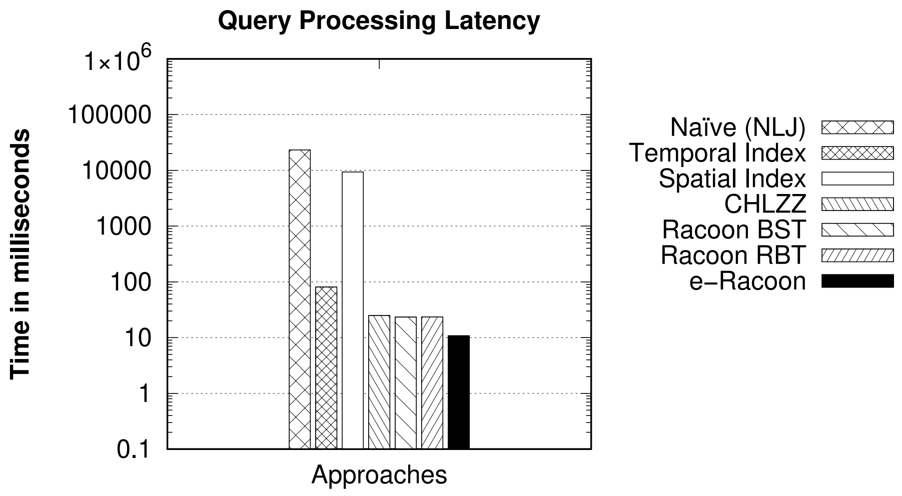

Figure 40: Query processing latency for datasets DS_7.

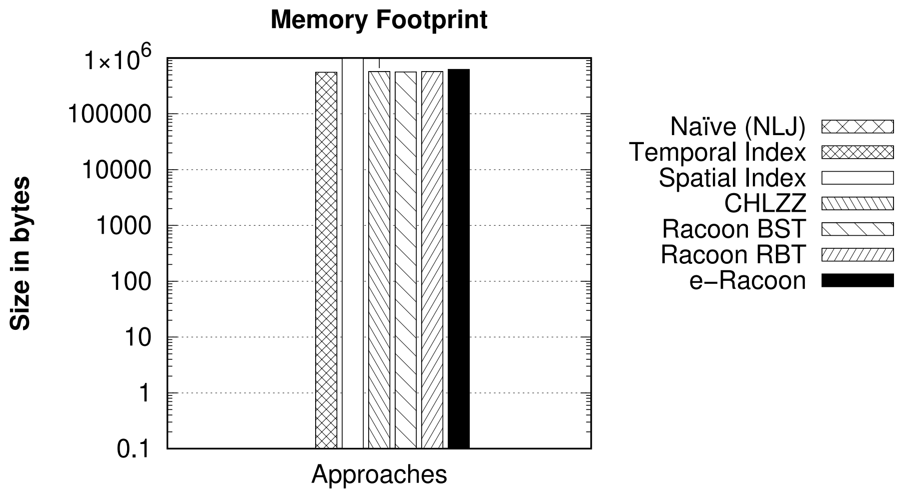

Figure 41: Memory footprint in bytes for datasets DS_8.

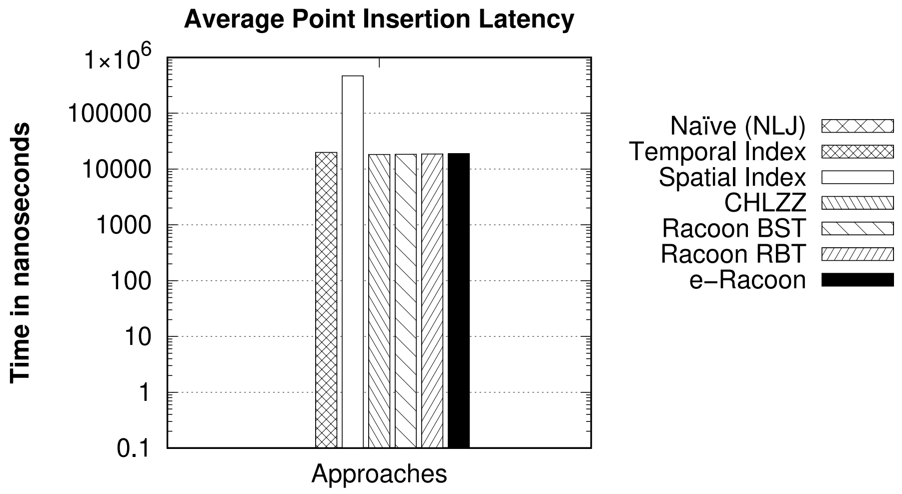

Figure 42: Average point insertion latency for datasets DS_8.

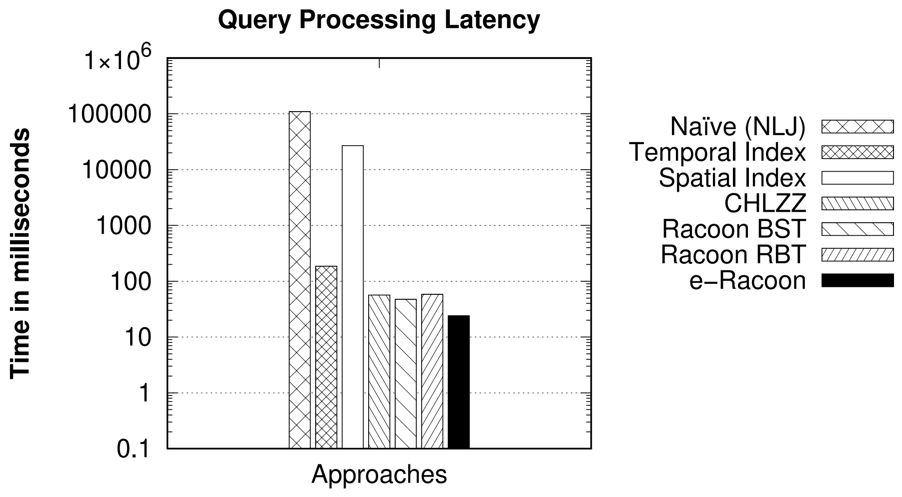

Figure 43: Query processing latency for datasets DS_8.

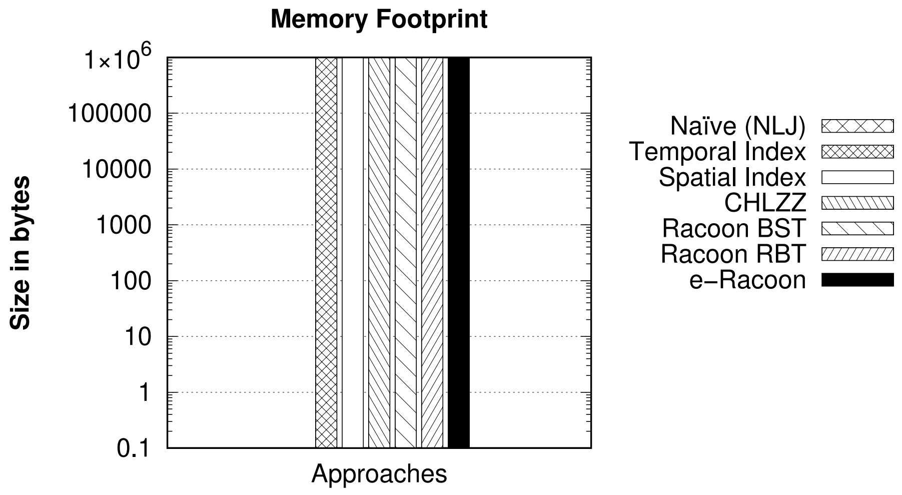

Figure 44: Memory footprint in bytes for datasets DS_9.

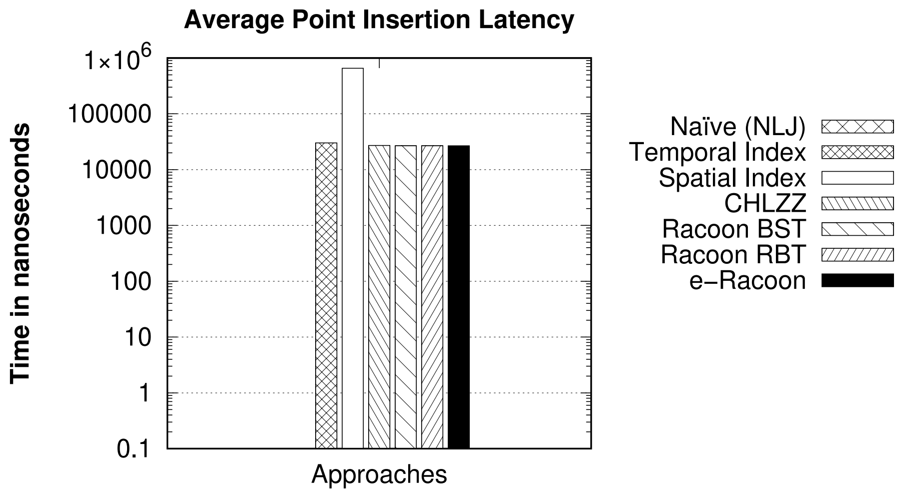

Figure 45: Average point insertion latency for datasets DS_9.

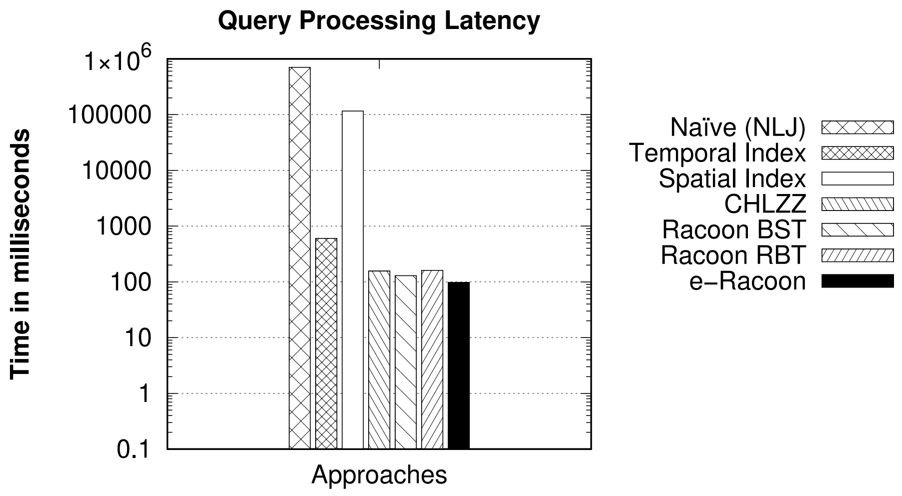

Figure 46: Query processing latency for datasets DS_9.

3.4.2.3 Experiment 3 (DS_7, DS_8 and DS_9)

DS_7 dataset is realistic, where MOs repeat some of the zones they occupy. The memory consumption is comparable across all approaches except the Spatial Index where timestamps are stored as is (no intervals). In processing latency, *e-Racoon* outperforms CHLZZ by a gain of more than 56% (CHLZZ 24.99ms and *e-Racoon* 10.86ms). This is a result of the temporal pointer optimization. It also outperforms CHLZZ by 300ns and the Spatial Index by orders of magnitude, since the Spatial Index will have oversize nodes of repeated zones with linear storage of timestamps. DS_8 and DS_9 exhibit similar results.

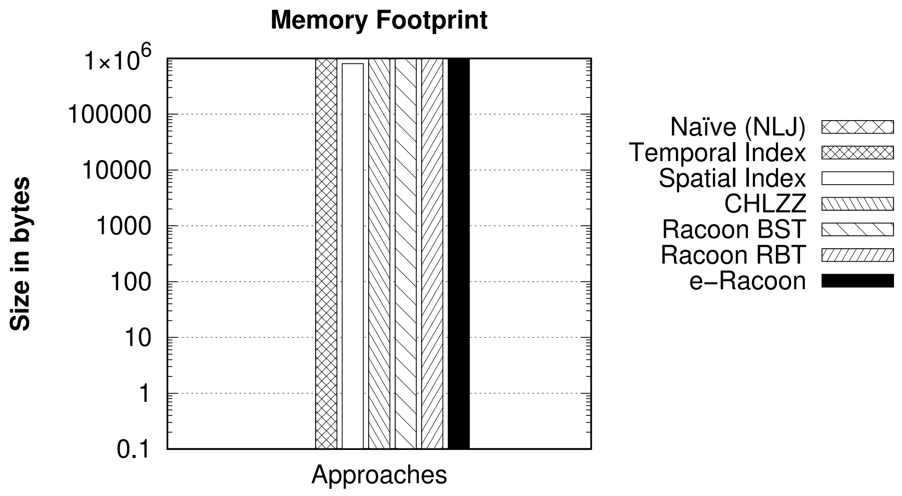

Figure 47: Memory footprint in bytes for datasets DS_4.

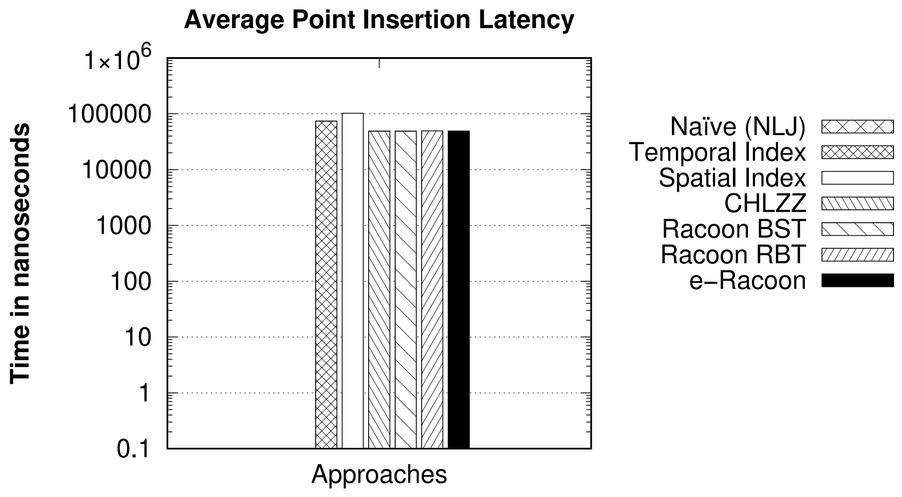

Figure 48: Average point insertion latency for datasets DS_4.

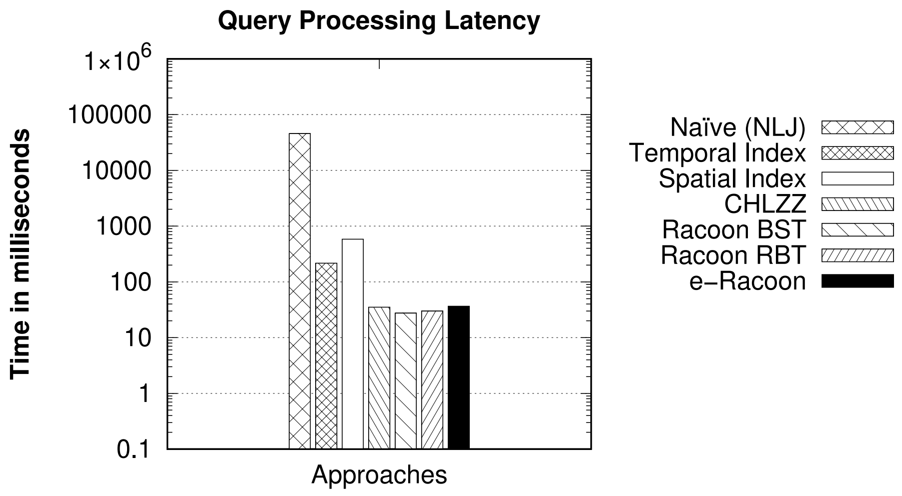

Figure 49: Query processing latency for datasets DS_4.

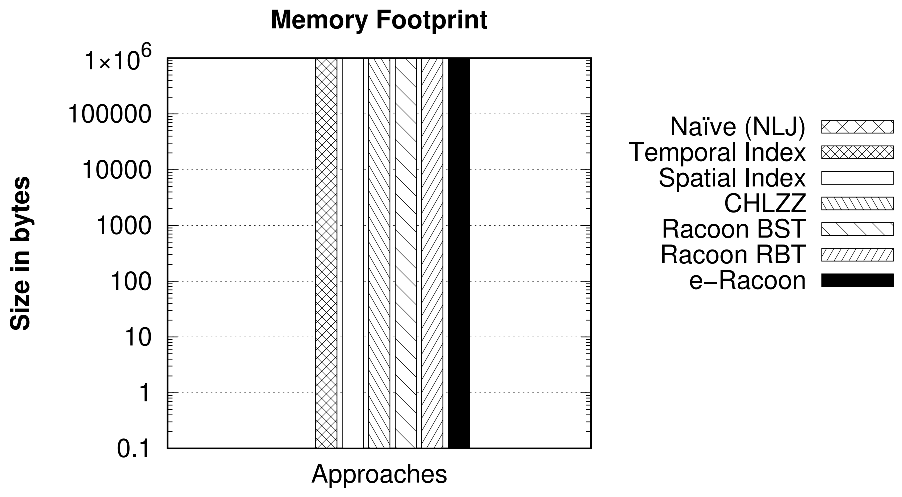

Figure 50: Memory footprint in bytes for datasets DS_5.

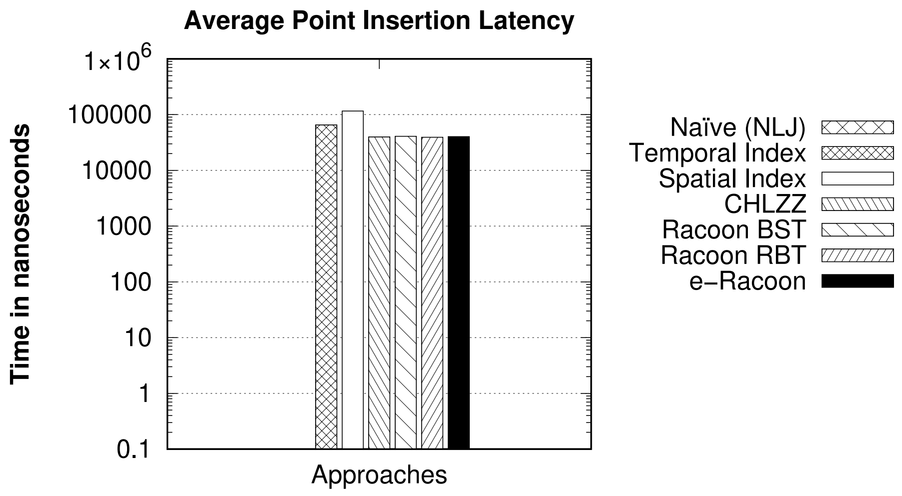

Figure 51: Average point insertion latency for datasets DS_5.

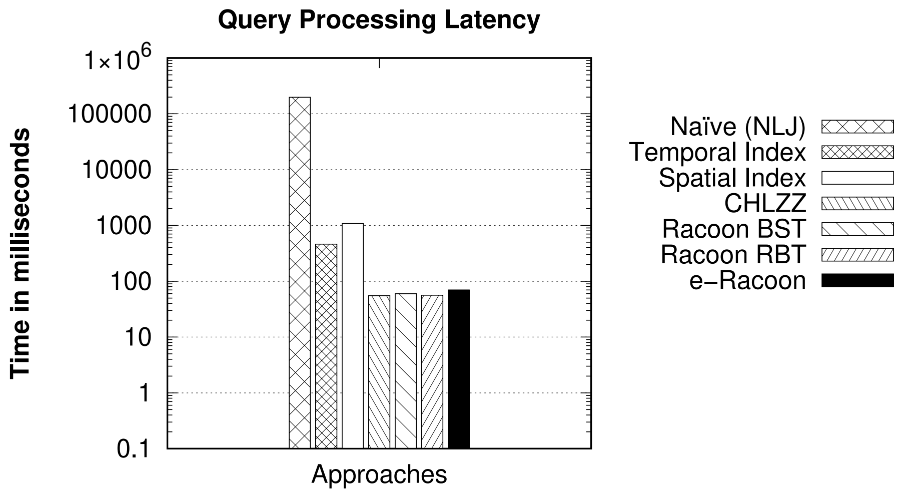

Figure 52: Query processing latency for datasets DS_5.

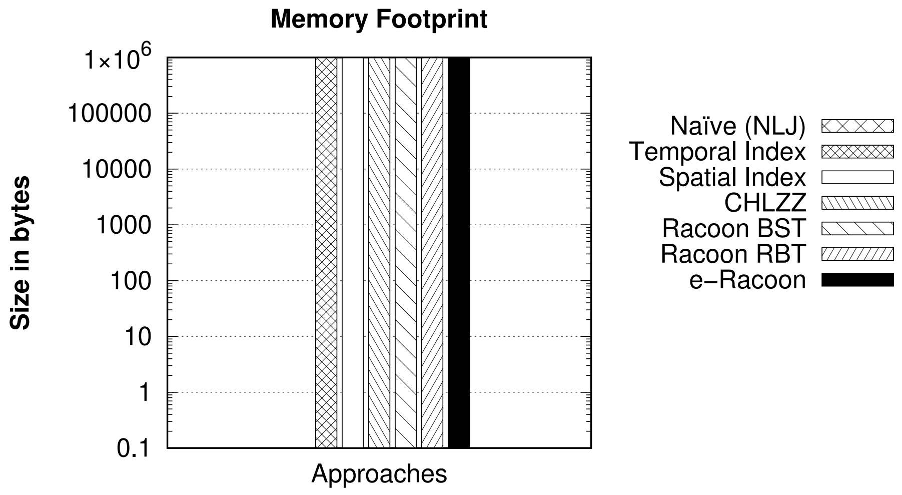

Figure 53: Memory footprint in bytes for datasets DS_6.

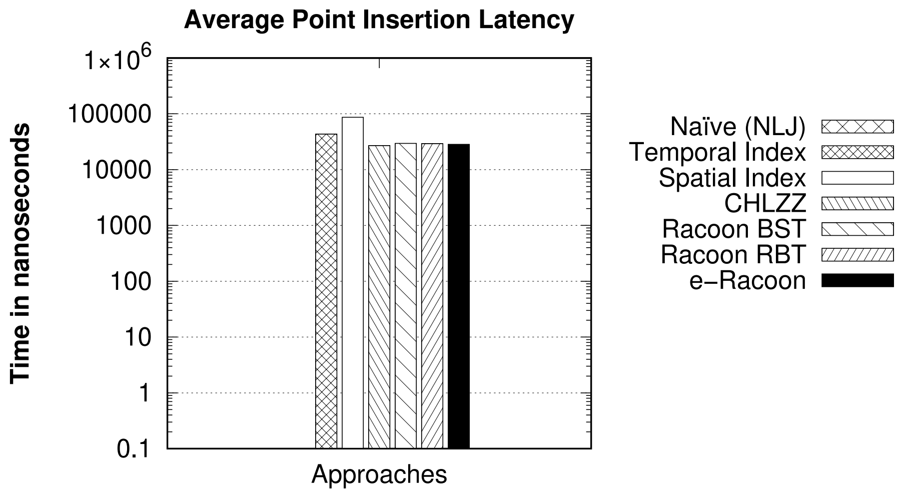

Figure 54: Average point insertion latency for datasets DS_6.

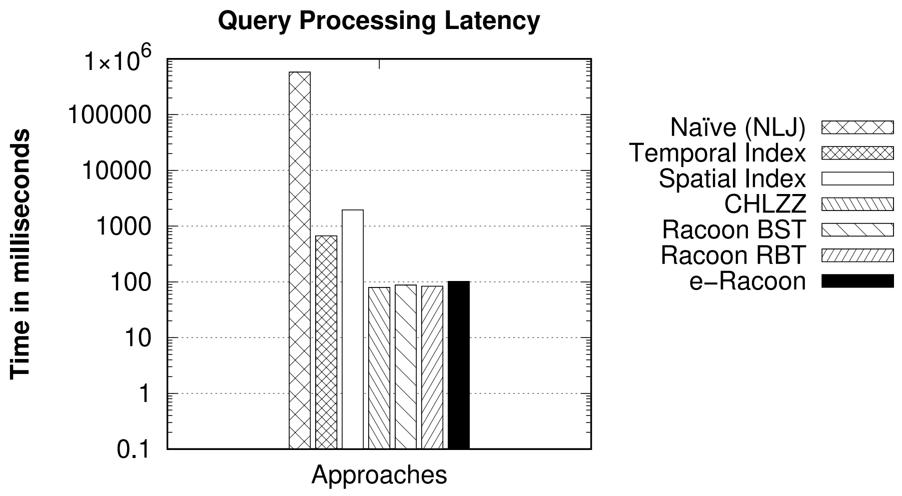

Figure 55: Query processing latency for datasets DS_6.

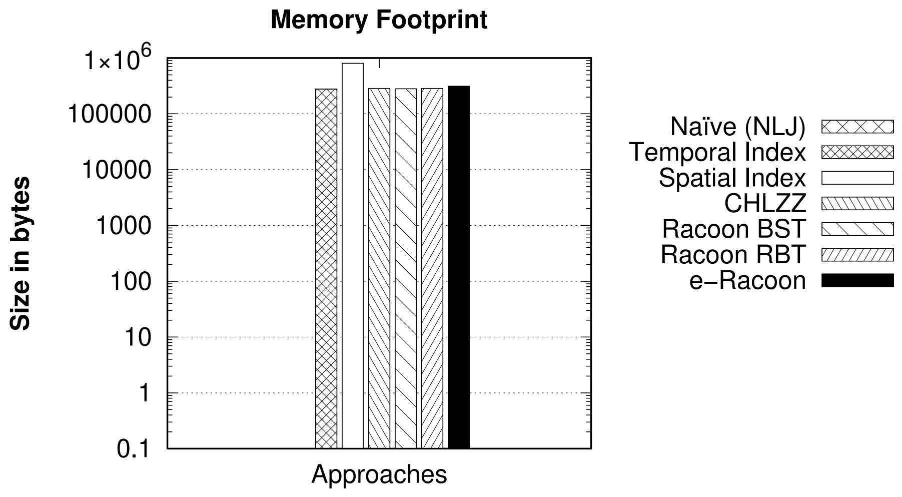

Figure 56: Memory footprint in bytes for datasets DS_10.

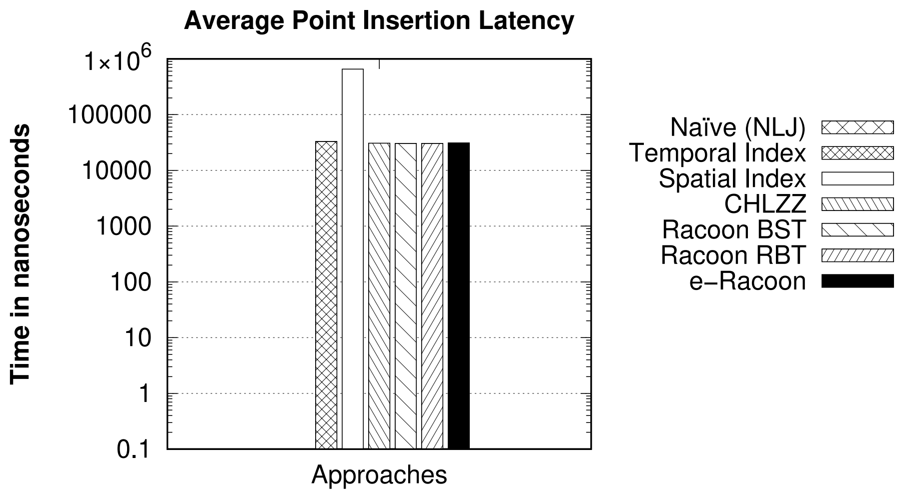

Figure 57: Average point insertion latency for datasets DS_10.

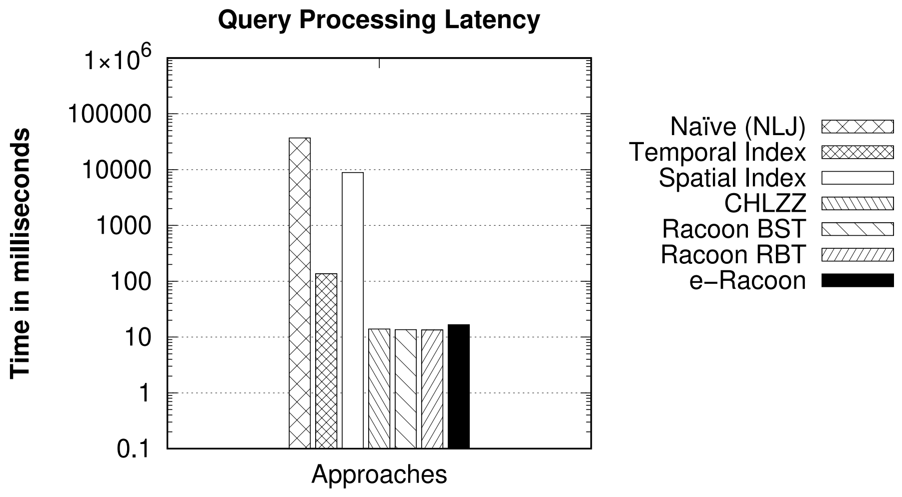

Figure 58: Query processing latency for datasets DS_10.

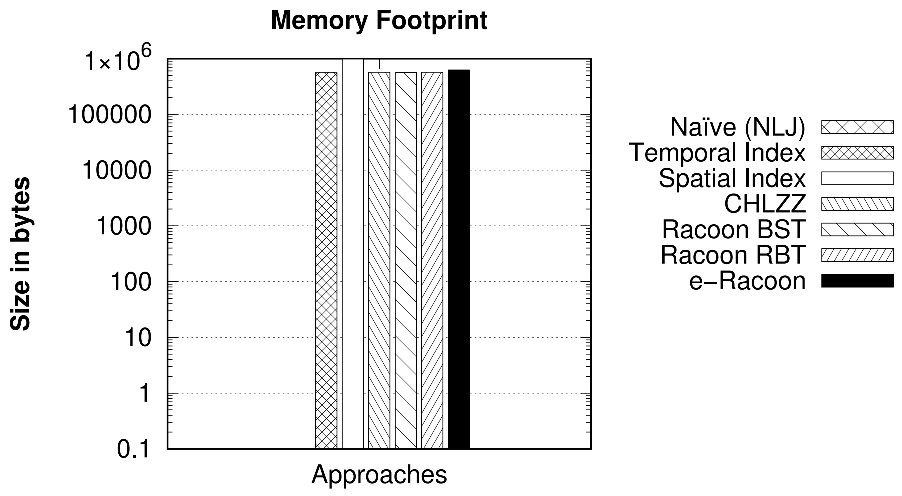

Figure 59: Memory footprint in bytes for datasets DS_11.

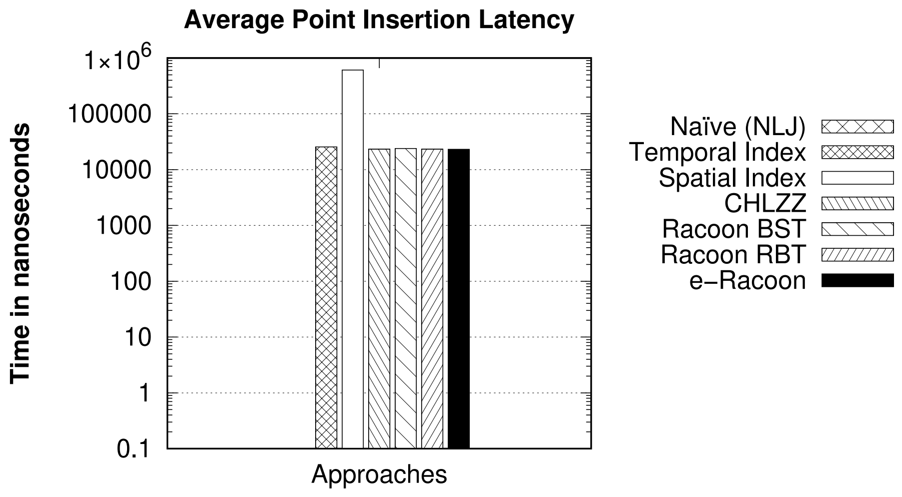

Figure 60: Average point insertion latency for datasets DS_11.

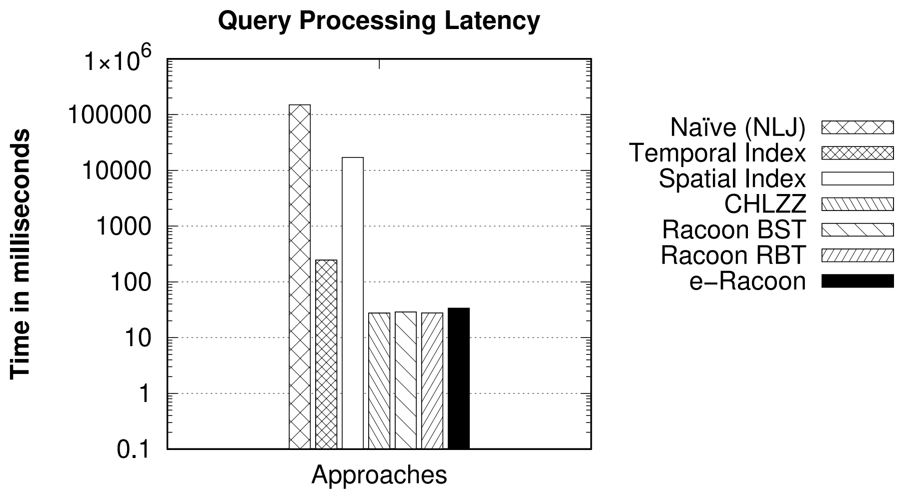

Figure 61: Query processing latency for datasets DS_11.

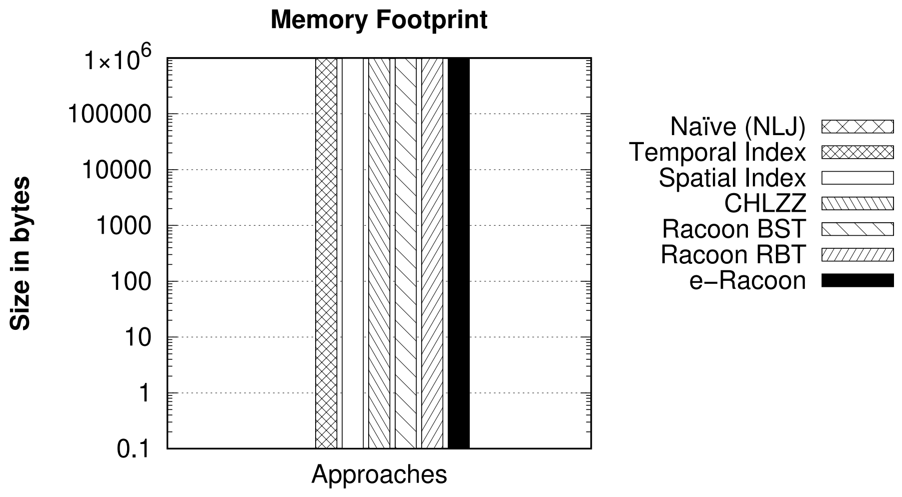

Figure 62: Memory footprint in bytes for datasets DS_12.

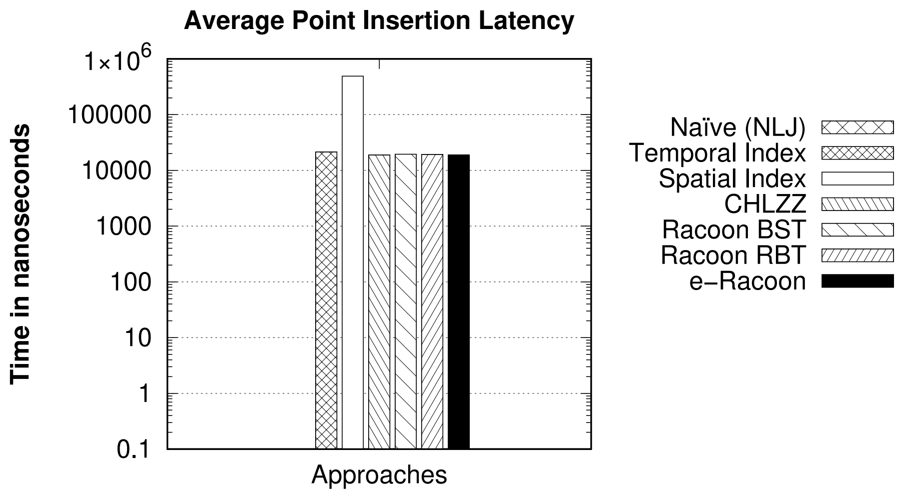

Figure 63: Average point insertion latency for datasets DS_12.

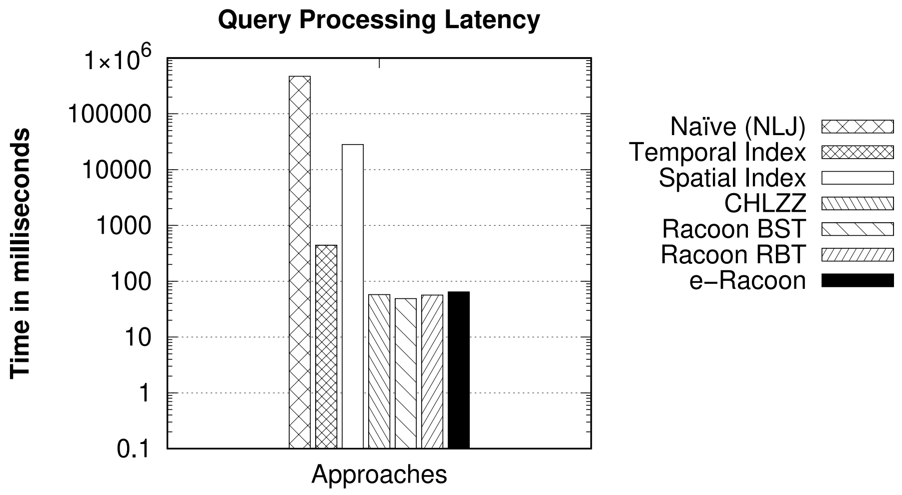

Figure 64: Query processing latency for datasets DS_12.

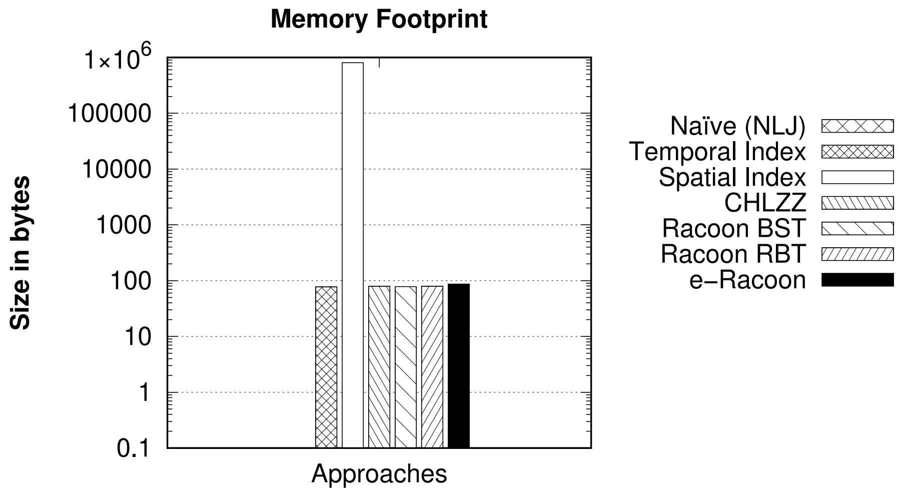

Figure 65: Memory footprint in bytes for datasets DS_16.

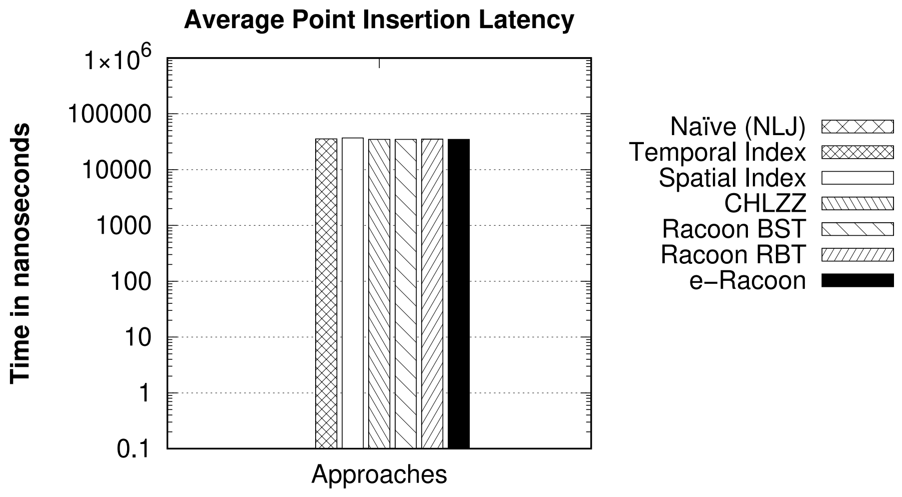

Figure 66: Average point insertion latency for datasets DS_16.

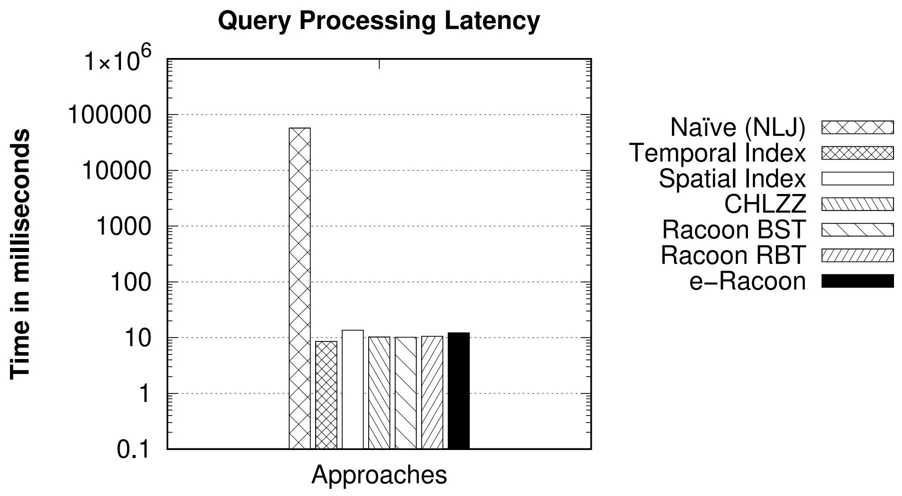

Figure 67: Query processing latency for datasets DS_16.

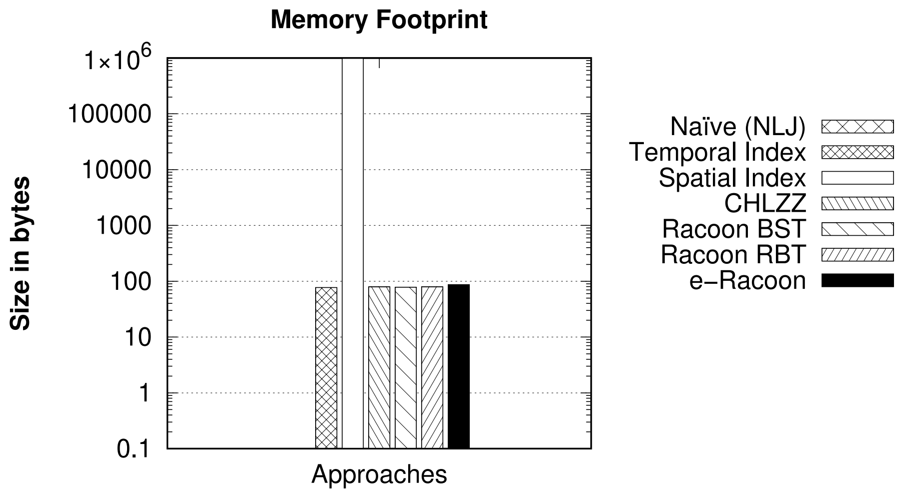

Figure 68: Memory footprint in bytes for datasets DS_17.

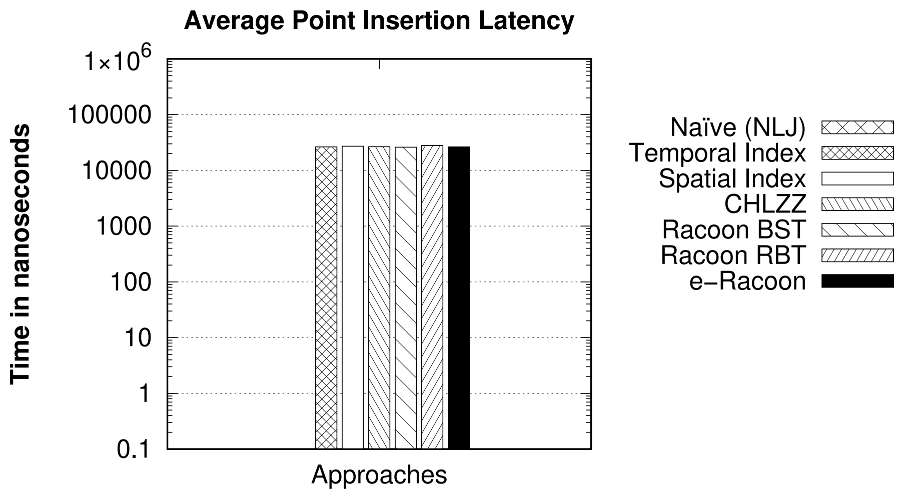

Figure 69: Average point insertion latency for datasets DS_17.

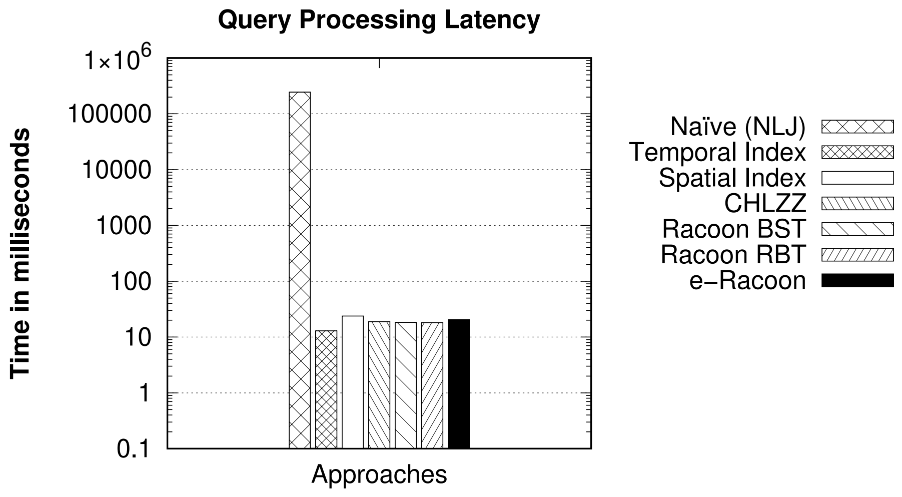

Figure 70: Query processing latency for datasets DS_17.

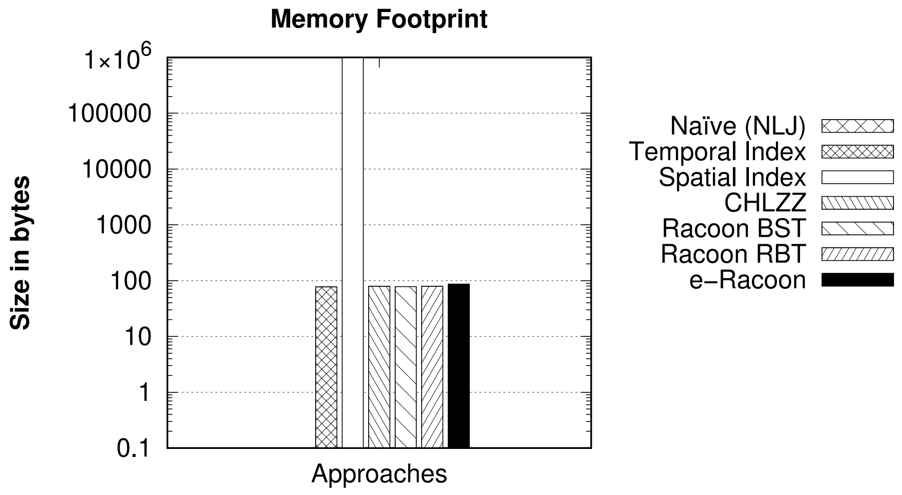

Figure 71: Memory footprint in bytes for datasets DS_18.

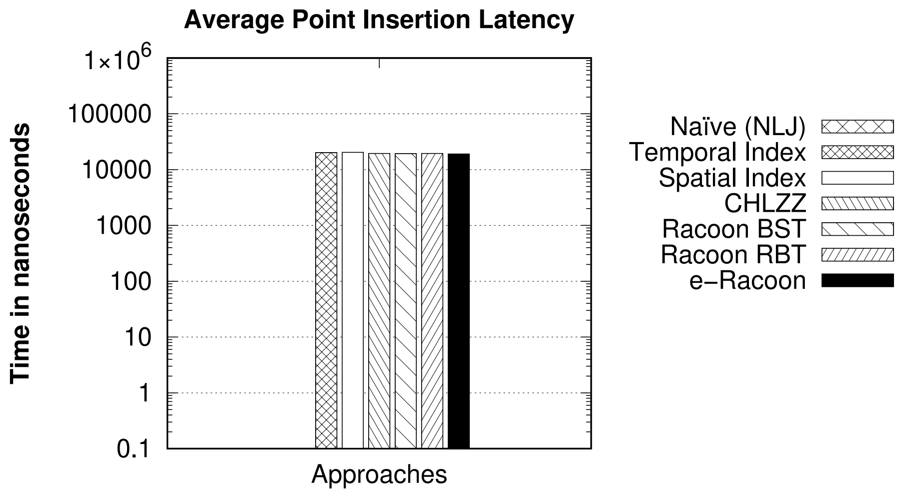

Figure 72: Average point insertion latency for datasets DS_18.

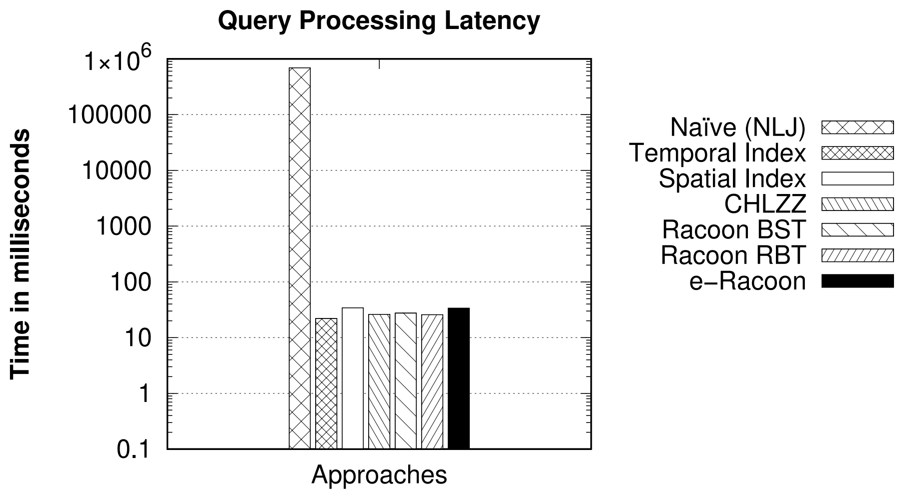

Figure 73: Query processing latency for datasets DS_18.

3.4.2.4 Experiment 4 (DS_4 - DS_6, DS_10 - DS_12, and DS_16 - DS_18)

These datasets exhibited similar results to the results of Experiment 3.4.2.3 in memory footprint and insertion time. However, they differ from Experiment 3.4.2.3 by having no contacts. This results in *e-Racoon* slightly under-performing compared to CHLZZ and *e-Racoon* variations, because in addition to stabbing to find contacts, the temporal pointer is accessed in anticipation that the next point in the trajectory will result in a contact.

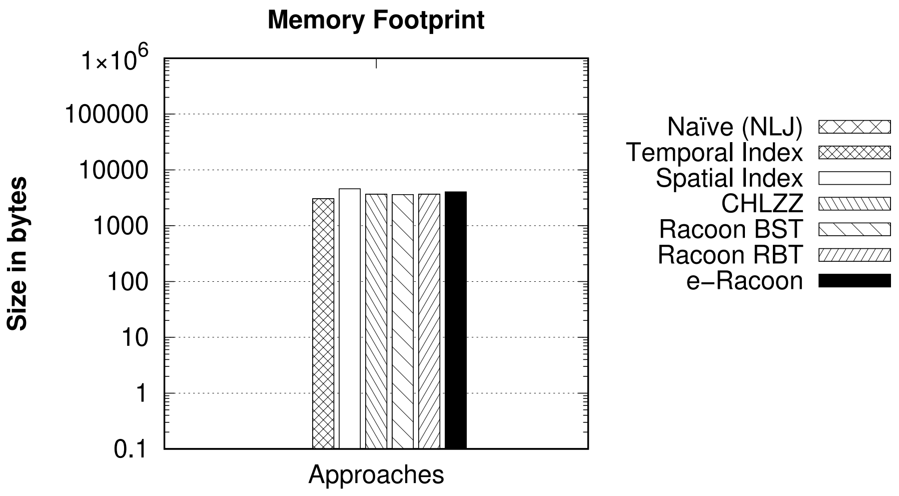

Figure 74: Memory footprint in bytes for datasets BJ_S.

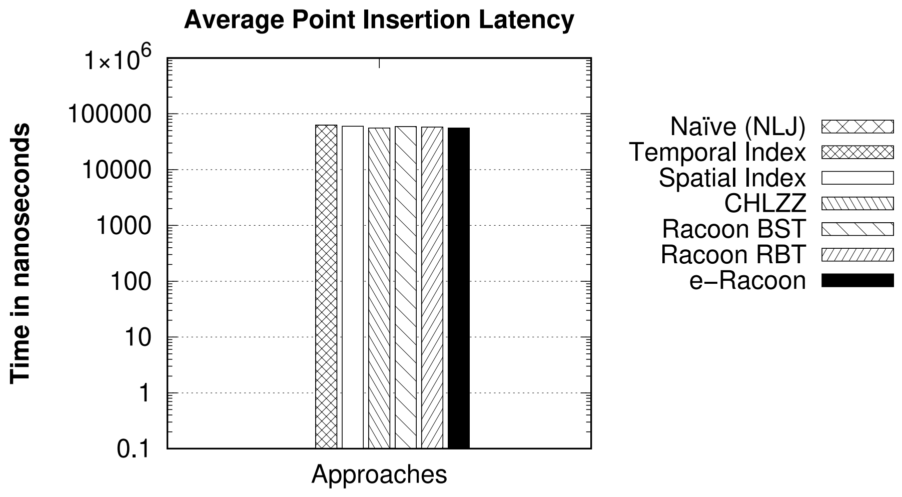

Figure 75: Average point insertion latency for datasets BJ_S.

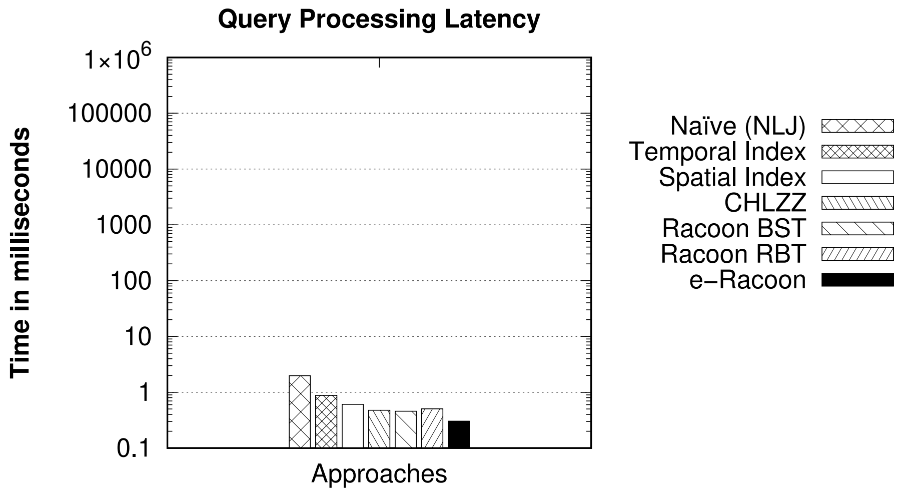

Figure 76: Query processing latency for datasets BJ_S.

3.4.2.5 Experiment 5 (BJ_S)

The dataset has an average characteristic, and we observe that *e-Racoon* outperformed all the other approaches, and outperformed CHLZZ by 37.5% (CHLZZ 0.48ms and *e-Racoon* 0.30ms). The point insertion is comparable for all approaches. The trade-off is the slightly higher memory footprint for storing the temporal pointer.

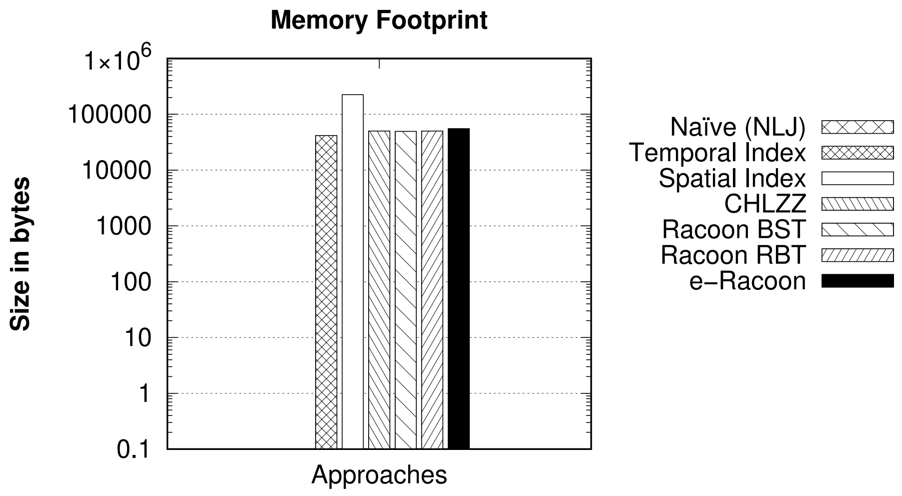

Figure 77: Memory footprint in bytes for datasets BJ_L.

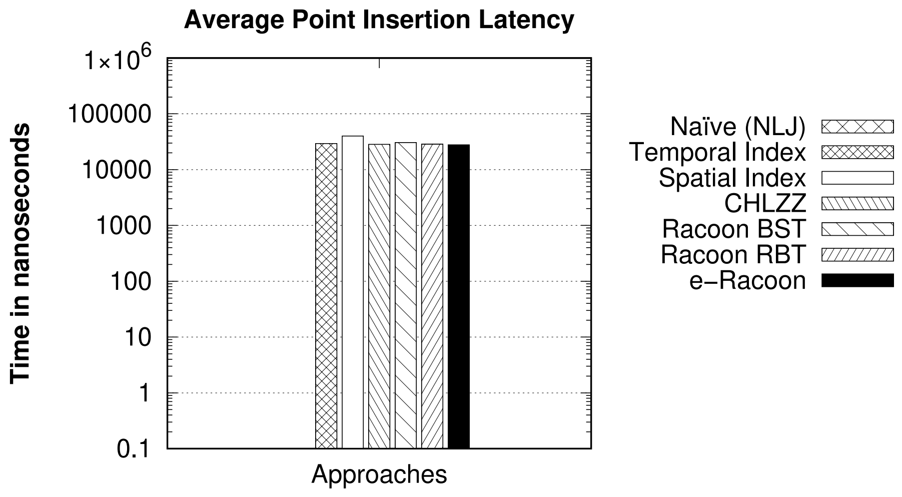

Figure 78: Average point insertion latency for datasets BJ_L.

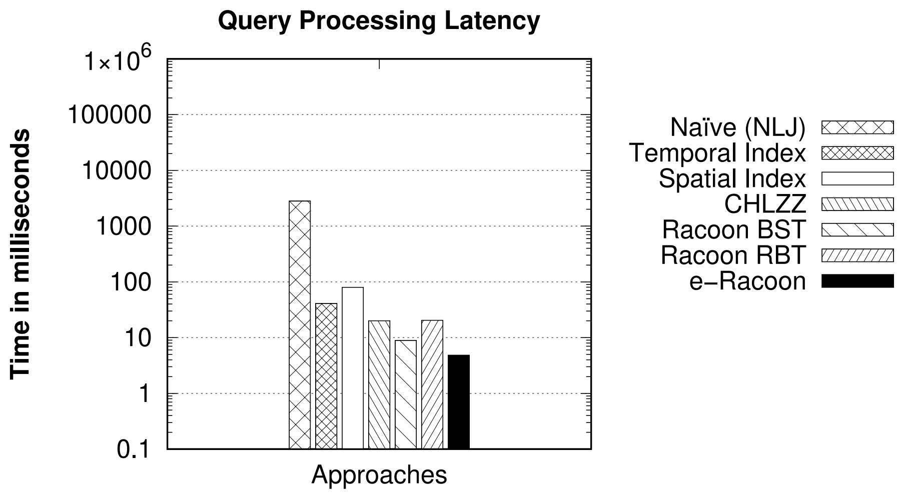

Figure 79: Query processing latency for datasets BJ_L.

3.4.2.6 Experiment 6 (BJ_L)

The dataset has an average characteristic as in BJ_S with longer trajectories. *e-Racoon* outperformed all the other approaches in processing latency, and specifically outperformed CHLZZ by more than 75% (CHLZZ 20.07ms and *e-Racoon* 4.84ms). The point insertion is comparable for all approaches, and the memory footprint is comparable for all approaches except the Spatial Index. We attribute this to the repetition of some zones which results in linear storage of timestamps. This is also clear in the processing latency for the Spatial Index.

3.4.2.7 Discussion

The experimental results show that *e-Racoon* outperforms the state-of-the-art approach (CHLZZ) [8] by more than 56% in realistic synthetic datasets and by up to 75% in real-world datasets at the cost of slightly higher memory footprint due to storing an extra temporal pointer per node in the tree structure.

The performance gain in the realistic synthetic datasets with full contacts ($\alpha = 1$ and $\beta = n$) is consistent with the complexity analysis discussion in Section 3.3.4, where the time complexity of *e-Racoon* is linear ($\mathcal{O}(n + \log(m))$). In addition, the synthetic datasets with no contacts ($\alpha = \beta = 0$) show that *e-Racoon*, its variations, and the the state-of-the-art approach (CHLZZ) exhibited comparable performance of linearithmic complexity ($\mathcal{O}(n \log(m))$) that is consistent with our complexity analysis discussion in Section 3.3.4.

Finally, *e-Racoon* should be the choice for processing TAJ queries in realizing our *Pri-DIER* and similar frameworks.

3.5 Related Work

Spatiotemporal joins are in the core of contact tracing (CT) and [27] concluded that specialized spatiotemporal joins perform better than spatial join followed by a temporal join and vice versa. Thus, we focus on specialized spatiotemporal joins related to CT.

P. Chao et al. [8] formulate a trajectory contact search query that aims at finding trajectories that are in contact directly and indirectly. They propose an iteration-based solution to answer their formulated query. They traverse the trajectory space in a chronological order looking for direct contacts using BFS or DFS. They also provide two optimizations to improve the time and space efficiency. First, they propose an algorithm that prunes the space of trajectories that need to be examined. Second, similar to us they use a spatial grid and interval indexing to build their own Interval Grid Index, that is different than ours. Their approach is also different than ours since they are not answering an aggregate query, and do not consider durations of contacts. Also, the solution is centralized where the trajectory of all users is stored in a single node.

In [57], X. Zhang et al. study the problem of finding contact similarity in uncertain trajectories. They coin their own contact similarity query and propose an index structure based on M-trees that stores approximate segments of trajectories and uses k nearest neighbor query to search for certain contact. In their approach they do not study contact tracing specifically, and their query is not an aggregate query that considers durations of contacts. Also, their approach is spatially neutral, while the focus in our work is for indoor trajectories for contact tracing purpose.

Similarly to [57], in [32], Liu et al. proposes a solution for contact tracing over uncertain trajectories in indoor environments. They define their own Indoor Contact Query (ICQ) that finds contact instances between moving objects based on a certain probability threshold. From those contacts that exceed a certain probability threshold, they calculate the duration of contact. The approach of their proposed solution is based in its core on a graph model along with B-tree indexing of all stored raw trajectories. We differ from [32] in our problem formulation and our approach. We do not use a probabilistic method, do not store raw trajectories in a centralized environment, and preserve the privacy of users.

REACT is a contact tracing system that is presented in [52]. The authors employ Geo-Indistinguishability which is based on differential privacy. Also, they give the choice to users to control the privacy levels. The authors employ R-trees as spatiotemporal index in their system as opposed to our combination of grid spatial index and interval trees. Our approach also differs from REACT in the privacy preserving technique by processing data locally.

To conclude, to the best of our knowledge, existing contact tracing work ignores the cumulative duration of multiple contacts and is hence unable to assess the exposure risk in higher resolution (e.g., [6, 15, 52, 8, 32]). Particularly, these works focus on discrete contacts with duration constraints on each individual contact incident instead of having a holistic view of the quanta accumulated over a period of time (e.g., two weeks) [14]. In contrast, our work formulates a specialized aggregate spatiotemporal join that can take into consideration the *cumulative* duration of the contacts for the purpose of viral risk assessment in a privacy preserving fashion.

3.6 Summary

In this chapter, we discussed *PriDIER*, our contact tracing framework for privately detecting indoor close contact and exposure risk across multiple users and from multiple sources. *PriDIER* processes the contact tracing query locally at the users devices to preserve their privacy.

In particular, we discussed the difference between the *Pull-based* and the *Push-based* protocols for contact tracing between the CA and MOs. Then we discussed the different components of the MO and the CA in details. Subsequently, we discussed the *temporal aggregation join* query which is at the core of processing the contact tracing queries at the *PriDIER* framework and presented *e-Racoon*, a novel in-memory access structure which optimizes processing the *temporal aggregation joins.*

e-Racoon combines spatial grid to stores zones that individual MOs occupy and Interval trees with nodes connected in a linked list that efficiently process trajectory joins and computes the duration of contacts cumulatively.

We evaluated *e-Racoon*'s efficiency both analytically and experimentally, and our experimental results show that *e-Racoon* outperforms the baselines and the state-of-the-art approaches used for contact tracing by more than 100% using synthetic datasets and by up to 75% using a real-world dataset.

Currently, the *PriDIER* solution considers users interacting in zones regardless of whether users wear masks or not. It is possible to enhance the Exposure Aggregator in MOs (sec. 3.2.4) with a function that quantifies the risk in a more accurate way via considering the mask factor (e.g., [37]).

4.0 Conclusions

In this chapter, we summarize our contributions, shed light on the future extensions and directions of the work presented, and we discuss the broader impact of this dissertation.

4.1 Summary of Contributions

In this dissertation we aim at optimizing temporally ordered data correlations and spatiotemporal data (i.e., trajectories) aggregate joins. These are often used in two categories of essential applications, namely health monitoring and contact tracing systems.

In health monitoring systems, live data streams that recently arrive need to be correlated with full precision. This is a computationally challenging task due to the significant delays encountered in the production of results. To this end, *our first hypothesis is that efficient and intelligent grouping and processing of pairs of data streams is required in a way that reduces the delay of producing results, and increases the task throughput according to the task goal.*

In contact tracing systems, processing large number of trajectory pairs for calculating the duration of the multiple contacts, and the cumulative viral exposure by different direct and indirect contacts from different sources is a computationally challenging and potentially privacy invasive task. To tackle that challenge, *our second hypothesis is that an aggregate spatiotemporal join query for contact tracing needs to be processed locally on users' mobile devices for maximizing their privacy while optimizing its processing.*

We addressed the above hypotheses by developing two frameworks:

- *Detection of Correlated Data Streams (DCS)* framework (Chapter 2): The framework intelligently schedules correlating pairs of data streams within a micro-batch and across micro-batches. Furthermore, we developed two novel algorithms that are utilized by the *DCS* framework. The first is ***iBRAID-DCS***, which explores the pairs in round robin fashion, and the second is ***PriCe-DCS***, which uses a priority function based on historical success rate, cost and PCC to schedule the pairs correlation task [41, 5]. Lastly, we coin nine different policies that our novel ***PriCe-DCS*** algorithm can employ when analyzing consecutive micro-batches. These policies can increase the efficiency when detecting correlated live data streams and/or address different exploration requirements.

- *Privately Detecting Indoors Exposure Risk (PriDIER)* framework (Chapter 3): *PriDIER* is a distributed framework for processing contact tracing that measures exposure risk across users in an effective and privacy-preserving manner. We developed two communication protocols, a push-based and a pull-based for trajectory transmission between the framework components. Those protocols minimize the exchanged data and reduce the power consumption at users' mobile devices at the cost of partial privacy violation of revealing MOs past locations. Furthermore, we develop and evaluate analytically and experimentally *e-Racoon*, a novel in-memory access structure that weaves spatial and temporal indexing to efficiently process ***temporal aggregation join*** queries at a users' mobile devices.

4.2 Future Work

The work in this dissertation can be extended in many directions, we mention few here:

- *Policy Self-tuning DCS* - In our current work, we choose the policy manually before the exploration task is started. One possibility is that after choosing the goal of the exploration task (i.e., exploration or exploitation), an active learning ML model employ the most efficient policy within the exploration category to maximize the accuracy and throughput and minimize the latency.
- *PriDIER Data Compression* - Given the *PriDIER* framework nature of processing data at the users mobile devices and the battery limitations of such devices, it is worth exploring and evaluating data compression techniques to further reduce the amount of data exchanged in the contact tracing protocol.
- *PriDIER Contact Tracing Protocol* - Extending the current Pull-based and Push-based protocols via studying the period at which a Trusted Central Authority (CA) is pushing

data or a Moving object (MO) pull data can facilitate detecting infections and exposure risks at higher rate.

- *Scalable PriDIER* - The current framework employs a single CA. This can be computationally challenging in case of large number of users. A scalabel distributed CA solution can enhance the user experience and increase the system efficiency.
- *Energy Evaluation PriDIER* - Due to battery restrictions at the MO devices, measuring and evaluating the energy efficiency is valuable. This can be achieved by either by simulation or by developing a prototype and measuring the the energy consumption of usage and communication.
- *Accuracy Aware PriDIER* - Developing techniques (e.g., considering mask wearing and distance) and incorporating existing tools (e.g., [37]) can help reducing inaccuracies and false positives detection of exposure risk and close contacts further.
- *PriDIER Complete System Implementation* - Implementing a complete contact tracing system based on the *PriDIER* framework would facilitate enhanced understanding of the environment at which the contact tracing is employed through inferring statistical and analytical data from the current framework components.

4.3 Broad Impact

This dissertation was motivated by two classes of spatiotemporal-based applications of great economic and social impact.

The first is health monitoring applications which are used widely in scientific and biomedical research and general business analytics, such as genome sequencing analysis, banking and e-commerce systems, and bridges monitoring systems. Typically these applications involve a tremendous amount of processing nodes that frequently face failures and down-time. Those failures and down-times result in significant financial losses.

The work in this dissertation optimizes the operations behind the analytics of monitoring the health of such processing nodes. The ability to detect signs of failures through correlating the behavior of such nodes to each other and being able to employ contingency plans on time before the failure occurrence can minimize such losses.

The second is contact tracing applications that are used in many analytics applications, such as infection detection systems, traffic control systems, crime control systems, ride sharing optimization systems. Those systems rely on detecting contacts among traced objects in the environment. For example, the COVID-19 pandemic has made a massive impact on humanity in many fronts, such as human health and world economy. The breakout of a new pandemic of respiratory diseases is currently a possible threat. Effectively detecting the early cases of a new breakout disease will significantly prevent normal day-to-day life disruptions. Achieving this will prevent economical and health losses. Thus, employing a dynamic, privacy centered, and effective solution for contact tracing and infection detection helps reducing the losses on all fronts.

The work in this dissertation can be employed by governments (e.g., [21, 22, 40]) and health organizations to enhance current contact tracing of respiratory diseases to effectively and accurately detect infections in an organizational and societal levels while acknowledging the participants data privacy rights.

Bibliography

[1] Walid G. Aref Ahmed R. Mahmood, Sri Punni. Spatio-temporal access methods: a survey (2010 - 2017). *GeoInformatica*, 23:1–36, 2019.

[2] Rakan Alseghayer. Racoon: Rapid contact tracing of moving objects using smart indexes. In *IEEE MDM*, 2021.

[3] Rakan Alseghayer, Daniel Petrov, and Panos K. Chrysanthis. Strategies for detection of correlated data streams. In *Proceedings of the 5th International Workshop on Exploratory Search in Databases and the Web*, 2018.

[4] Rakan Alseghayer, Daniel Petrov, Panos K. Chrysanthis, Mohamed Sharaf, and Alexandros Labrinidis. Detection of highly correlated live data streams. In *Proceedings of the International Workshop on Real-Time Business Intelligence and Analytics*, 2017.

[5] Rakan Alseghayer, Daniel Petrov, Panos K. Chrysanthis, Mohamed Sharaf, and Alexandros Labrinidis. Dcs: A policy framework for the detection of correlated data streams. In *Real-Time Business Intelligence and Analytics*, pages 191–210, 2019.

[6] Thamer Altuwaiyan, Mohammad Hadian, and Xiaohui Liang. Epic: Efficient privacy-preserving contact tracing for infection detection. In *IEEE ICC*, 2018.

[7] Shivnath Babu, Utkarsh Srivastava, and Jennifer Widom. Exploiting k-constraints to reduce memory overhead in continuous queries over data streams. *ACM Trans. Database Syst.*, 29(3):545–580, 2004.

[8] Pingfu Chao, Dan He, Lei Li, Mengxuan Zhang, and Xiaofang Zhou. Efficient trajectory contact query processing. In Christian S. Jensen, Ee-Peng Lim, De-Nian Yang, Wang-Chien Lee, Vincent S. Tseng, Vana Kalogeraki, Jen-Wei Huang, and Chih-Ya Shen, editors, *Database Systems for Advanced Applications*, pages 658–666, Cham, 2021. Springer International Publishing.

[9] Adam Charane, Matteo Ceccarello, Anton Dignös, and Johann Gamper. Efficient computation of all-window length correlations. In *Digital Business and Intelligent Systems*, pages 251–266, 2022.

[10] Richard Cole, Dennis Shasha, and Xiaojian Zhao. Fast window correlations over uncooperative time series. In *ACM SIGKDD*, pages 743–749, 2005.

[11] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to Algorithms.* 4th edition, 2022.

[12] Constantinos Costa, Xiaoyu Ge, and Panos K. Chrysanthis. Caprio: Context aware path recommendation exploiting indoor and outdoor information. In *2019 20th IEEE International Conference on Mobile Data Management (MDM)*, pages 431–436, 2019.

[13] Constantinos Costa, Xiaoyu Ge, and Panos K. Chrysanthis. Caprio: Graph-based integration of indoor and outdoor data for path discovery. In *Proceedings of the 45th International Conference on Very Large Data Bases*, pages 1878–1881, 2019.

[14] Constantinos Costa, Brian T. Nixon, Sayantani Bhattacharjee, Benjamin Graybill, Demetrios Zeinalipour-Yazti, Walter Schneider, and Panos K. Chrysanthis. A context, location and preference-aware system for safe pedestrian mobility. In *2021 22nd IEEE International Conference on Mobile Data Management (MDM)*, pages 217–224, 2021.

[15] Michael O. Cruz, Hendrik Macedo, and Adolfo Guimarães. Grouping similar trajectories for carpooling purposes. In *BRACIS*, 2015.

[16] Luping Ding, Nishant Mehta, Elke A. Rundensteiner, and George T. Heineman. Joining punctuated streams. In Elisa Bertino, Stavros Christodoulakis, Dimitris Plexousakis, Vassilis Christophides, Manolis Koubarakis, Klemens Böhm, and Elena Ferrari, editors, *Advances in Database Technology*, pages 587–604, 2004.

[17] Paul C. Erwin and Ross C. Brownson. *Principles of Public Health Practice.* Cengage Learning, 2016.

[18] Kaiyu Feng, Gao Cong, Sourav S. Bhowmick, Wen-Chih Peng, and Chunyan Miao. Towards best region search for data exploration. ACM SIGMOD, pages 1055–1070, 2016.

[19] The US Centers for Disease Control and Prevention. Covid-19 close contact. `https://www.cdc.gov/coronavirus/2019-ncov/daily-life-coping/determine-close-contacts.html`.

[20] Thriyambakam Krishnan Geoffrey J. McLachlan. *The EM Algorithm and Extensions.* John Wiley and Sons, 2008.

[21] Australia Gov. Covidsafe app. `https://www.health.gov.au/resources/apps-and-tools/covidsafe-app`.

[22] Singapore Gov. Tracetogether. `https://www.gov.sg/article/help-speed-up-contact-tracing-with-tracetogether`.

[23] Zhenwen He, Chonglong Wu, Gang Liu, Zufang Zheng, and Yiping Tian. Decomposition tree: A spatio-temporal indexing method for movement big data. *Cluster Computing*, 18:1481–1492, 2015.

[24] Stratos Idreos, Olga Papaemmanouil, and Surajit Chaudhuri. Overview of data exploration techniques. ACM SIGMOD, pages 277–281, 2015.

[25] Yahoo Inc. Yahoo finance historical data, 2016.

[26] Dimitrije Jankov, Sourav Sikdar, Rohan Mukherjee, Kia Teymourian, and Chris Jermaine. Real-time high performance anomaly detection over data streams: Grand challenge. ACM DEBS, pages 292–297, 2017.

[27] Seung-Hyun Jeong, Norman W. Paton, Alvaro A. A. Fernandes, and Tony Griffiths. An experimental performance evaluation of spatio-temporal join strategies. *Transactions in GIS*, 9(2):129–156, 2005.

[28] Alexander Kalinin, Ugur Cetintemel, and Stan Zdonik. Interactive data exploration using semantic windows. ACM SIGMOD, pages 505–516, 2014.

[29] Alexander Kalinin, Ugur Cetintemel, and Stan Zdonik. Searchlight: Enabling integrated search and exploration over large multidimensional data. *PVLDB*, pages 1094–1105, 2015.

[30] Dongeun Lee, Alex Sim, Jaesik Choi, and Kesheng Wu. Novel data reduction based on statistical similarity. ACM SSDBM, pages 21:1–21:12, 2016.

[31] Jin Li, David Maier, Kristin Tufte, Vassilis Papadimos, and Peter A. Tucker. Semantics and evaluation techniques for window aggregates in data streams. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, page 311–322, 2005.

[32] Tiantian Liu, Huan Li, Hua Lu, Muhammad Aamir Cheema, and Harry Kai-Ho Chan. Contact tracing over uncertain indoor positioning data. *IEEE TKDE*, pages 1–14, 2023.

[33] Katsiaryna Mirylenka, Michele Dallachiesa, and Themis Palpanas. Data series similarity using correlation-aware measures. ACM SSDBM, pages 11:1–11:12, 2017.

[34] Abdullah Mueen, Suman Nath, and Jie Liu. Fast approximate correlation for massive time-series data. ACM SIGMOD, pages 171–182, 2010.

[35] Mashaal Musleh, Mohamed F. Mokbel, and Sofiane Abbar. Let's speak trajectories. In *SIGSPATIAL*, 2022.

[36] Brian Nixon, Rakan Alseghayer, Constantinos Costa, Benjamin Graybill, Xiaozhong Zhang, and Panos K. Chrysanthis. Efficient detection of covid-19 exposure. In *IEEE MDM*, 2022.

[37] Brian T Nixon, Sayantani Bhattacharjee, Benjamin Graybill, Constantinos Costa, Sudhir Pathak, Walter Schneider, and Panos K Chrysanthis. Healthdist: a context, location and preference-aware system for safe navigation. In *2021 22nd IEEE International Conference on Mobile Data Management (MDM)*, pages 250–253. IEEE, 2021.

[38] Mahsa Orang and Nematollaah Shiri. Improving performance of similarity measures for uncertain time series using preprocessing techniques. ACM SSDBM, pages 31:1–31:12, 2015.

[39] M. H. Overmars. *The design of dynamic data structures*, volume 3204 of *Lecture notes in Computer Assisted Diagnosis*. Springer, Berlin, Heidelberg, 1983.

[40] Y. Park, Y. Choe, O. Park, and et al. Contact tracing during coronavirus disease outbreak, south korea, 2020. *Emerging Infectious Diseases*, 26:2465–2468, 2020.

[41] Daniel Petrov, Rakan Alseghayer, Mohamed Sharaf, Panos K. Chrysanthis, and Alexandros Labrinidis. Interactive exploration of correlated time series. In *Proceedings of the ExploreDB'17*, 2017.

[42] Dieter Pfoser, Christian S. Jensen, and Yannis Theodoridis. Novel approaches in query processing for moving object trajectories. In *Proceedings of the 26th International Conference on Very Large Data Bases*, page 395–406, 2000.

[43] Donald B. Rubin. Multiple imputation after 18+ years. *Journal of the American Statistical Association*, 91(434):473–489, 1996.

[44] Yasushi Sakurai, Spiros Papadimitriou, and Christos Faloutsos. Braid: Stream mining through group lag correlations. ACM SIGMOD, pages 599–610, 2005.

[45] Jens M. Schmidt. Interval stabbing problems in small integer ranges. In *Algorithms and Computation*, pages 163–172, 2009.

[46] Ilari Shafer, Kai Ren, Vishnu Naresh Boddeti, Yoshihisa Abe, Gregory R. Ganger, and Christos Faloutsos. Rainmon: An integrated approach to mining bursty timeseries monitoring data. ACM SIGKDD, pages 1158–1166, 2012.

[47] Zeyuan Shang, Guoliang Li, and Zhifeng Bao. DITA: distributed in-memory trajectory analytics. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference*, pages 725–740, 2018.

[48] Anatoli U Shein and Panos K Chrysanthis. Multi-query optimization of incrementally evaluated sliding-window aggregations. *IEEE Transactions on Knowledge and Data Engineering*, 34(8):3899–3911, 2020.

[49] Han Su, Shuncheng Liu, Bolong Zheng, Xiaofang Zhou, and Kai Zheng. A survey of trajectory distance measures and performance evaluation. *VLDB J.*, 29(1):3–32, 2020.

[50] Olga Troyanskaya, Michael Cantor, Gavin Sherlock, Pat Brown, Trevor Hastie, Robert Tibshirani, David Botstein, and Russ B. Altman. Missing value estimation methods for dna microarrays. *Bioinformatics*, 17(6):520–525, 06 2001.

[51] Lionel Sujay Vailshery. Forecast end-user spending on iot solutions worldwide from 2017 to 2025.

[52] Li Xiong, Cyrus Shahabi, Yanan Da, Ritesh Ahuja, Vicki Hertzberg, Lance Waller, Xiaoqian Jiang, and Amy Franklin. React: Real-time contact tracing and risk monitoring using privacy-enhanced mobile tracking. *SIGSPATIAL Special*, 12(2):3–14, October 2020.

[53] Jing Yuan, Yu Zheng, Xing Xie, and Guangzhong Sun. Driving with knowledge from the physical world. In *ACM SIGKDD*, page 316–324, 2011.

[54] Jing Yuan, Yu Zheng, Xing Xie, and Guangzhong Sun. T-drive: Enhancing driving directions with taxi drivers' intelligence. *IEEE TKDE*, 25(1):220–232, 2013.

[55] Eleni Tzirita Zacharatou, Farhan Tauheedz, Thomas Heinis, and Anastasia Ailamaki. Rubik: Efficient threshold queries on massive time series. ACM SSDBM, pages 18:1–18:12, 2015.

[56] Demetrios Zeinalipour-Yazti, Christos Laoudias, Constantinos Costa, Michail Vlachos, Maria I. Andreou, and Dimitrios Gunopulos. Crowdsourced trace similarity with smartphones. *IEEE Trans. Knowl. Data Eng.*, 25(6), 2013.

[57] Xichen Zhang, Suprio Ray, Farzaneh Shoeleh, and Rongxing Lu. Efficient contact similarity query over uncertain trajectories. In Yannis Velegrakis, Demetris Zeinalipour-Yazti, Panos K. Chrysanthis, and Francesco Guerra, editors, *Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021*, pages 403–408. OpenProceedings.org, 2021.

[58] Yunyue Zhu and Dennis Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *Proceedings of the 28th International Conference on Very Large Data Bases*, PVLDB, pages 358–369, 2002.

[59] Kostas Zoumpatianos, Stratos Idreos, and Themis Palpanas. Indexing for interactive exploration of big data series. ACM SIGMOD, pages 1555–1566, 2014.