SCALING GEOMETRIC MONITORING OVER DISTRIBUTED STREAMS

by

Alexandros D. Keros

A thesis submitted in partial fulfillment
of the requirements for the degree

of

UNDERGRADUATE

in

Electronic and Computer Engineering

Approved:

Dr. Vasilis Samoladas Major Professor	first reader Committee Member
second reader Committee Member	dean

Technical University of Crete
Chania, Crete, Greece

2015

Scaling Geometric Monitoring over Distributed Streams

by

Alexandros D. Keros, Undergraduate

Technical University of Crete, 2015

Abstract

BLAH BLAH

Thesis Supervisor: Dr. Vasilis Samoladas
Department: Electronic and Computer Engineering

(70 pages)

Public Abstract

BLAH BLAH

Acknowledgments

my mum

Contents

List of Tables

Table Page

List of Figures

Figure Page

Part I

INTRODUCTION AND PRELIMINARIES

Chapter 1

Introduction

1.1 Overview

A multitude of recent emergent applications require real-time handling of rapidly incoming data, that may as well be great in size and distributed in nature. Such applications that follow a continuous distributed monitoring model are classified as *Data Stream Systems* [1]. Notable examples include, among others, distributed sensors, ISP network traffic monitoring, telecommunication system management, event monitoring, and real-time analysis tools for financial data.

These systems differ from the traditional Database Management Systems (DBMS), in the sense that they are following a *pull paradigm*, where large scale event monitoring is required or continuous queries are issued, instead of the *push paradigm*, where one-shot queries normally take place. Data Stream Systems are required to efficiently process, in real-time, data streams that are of high volume, continuous, size unbound, and most likely violative, in the sense that it would be inefficient to store them in memory. Additionally, the distributed nature of some applications incur an additional challenge to such systems, for they are required to communicate via a bandwidth-limited and possibly delay-inducing network in order to synchronize, reorganize, and provide a real-time overview of the results.

Consequently, intelligent algorithms must be devised that are able to guarantee high accuracy standards while limiting the communication overhead induced to the distributed setting. Approaches such as collecting all data to a central node for processing and polling nodes for data updates, as easily implementable as they may be, are prohibitive is such a decentralized scenario, either due to the communication and storage overhead they induce, or due to the accuracy degradation they inflict.

The geometric threshold monitoring method proposed by Sharfman et al. [2] consists of a

geometric approach in which convex optimization theory is being employed in order to guarantee that communication between distributed nodes takes place only when needed, while maintaining strict bounds on the accuracy of the monitoring task. By decomposing the task to local constraints at the nodes, and by employing a clever mechanism for resolving constraint violations that are not representative of the system's state, i.e. false alarms, the communication between sites is significantly reduced without any accuracy degradation.

Following this framework, much work has emerged attempting to improve the communication bounds of the geometric monitoring method and to generalize the method to applications in the likes of continuous query answering. Following this trend, this thesis employs heuristics structured on top of multi-objective optimization problems and signal processing filters in order to better the performance of the geometric monitoring algorithm. Additionally, an existing method for hierarchical clustering of distributed streams, found in [15], is improved and simplified.

Evaluation of the proposed algorithms over synthetic and real-world datasets exhibits an improvement of up to 60% over the original geometric monitoring method. Furthermore, the behavior of the proposed algorithms in different settings is being researched.

1.2 Motivation

A lot of work towards this direction, based on GM. We believe that there is still room for improvements regarding the way the method handles and represents data streams

1.3 Contributions

This thesis contributes to the research on distributed streams following the geometric monitoring framework by providing:

- a heuristic method for resolving false threshold violations by employing multi-objective optimization theory and estimations of data stream moments in order to optimally position vectors representing data streams in space,
- an improved algorithm over an existing hierarchical node clustering method for deterministic violation resolution between a subset of distributed streams,

- a throughout evaluation of the proposed algorithms on synthetic and real-world data by employing the seminar geometric monitoring method as the null model. By implementing the aforementioned algorithms a significant reduction of the communication overhead induced by the geometric monitoring can be achieved. Furthermore, the cases where the proposed algorithms do not perform well are examined and the factors that hamper their performance are analyzed.

1.4 Thesis Outline

Chapter 2 provides the necessary theoretical background used throughout this thesis, including the geometric monitoring framework, multi-objective optimization theory, graph matching theory and the Savitzky-Golay smoothing and differentiating filter. Related work is surveyed in Chapter 3. The problem formulation, along with a detailed analysis of the implementation of the proposed methods follows in Chapters 4 and 5, respectively. Finally, experimental evaluation of our work takes place in Chapter 6, before providing concluding remarks, as well as proposals for future work, in Chapter 7.

Chapter 2

Theoretical Background

The present chapter contains the background knowledge required throughout the length of this thesis. Section 2.1 describes the framework of the *Geometric Monitoring method*. Section 2.2 presents *multi-objective optimization* and dives into the algorithms used in our implementation. Section 2.4 discusses *graph maximum weight matching*, and, finally, in Section 2.3 we explain the *Savitzky-Golay filtering* used for smoothing, velocity and acceleration approximation.

2.1 Geometric Monitoring of Distributed Streams

The *Geometric Monitoring* method [2] was devised as a way to monitor threshold crossings of arbitrary functions over distributed data streams, i.e. be able to determine whether an arbitrary *monitoring function* $f(\cdot)$ over the data streams violated a predetermined threshold ($f(\cdot) > T$ or $f(\cdot) < T$). By mapping data streams to a feature space defined by the dimensionality of each data stream update and monitoring the convex hull surrounding the value of the monitoring function, Sharfman et al. were able to decompose the monitoring task into local constraints and apply distributed threshold monitoring, while reducing the communication costs required by central data processing.

In the current section a detailed presentation of this method is taking place. In Subsection 2.1.1 two system architectures are shown, a decentralized scenario and a centralized one, where Geometric Monitoring can be applied. Subsection 2.1.2 explains the computational model, followed by the method's geometric interpretation in Subsection 2.1.3. Finally, in Subsection 2.1.4 the protocol implementing the Geometric Monitoring method is described.

2.1.1 System Architecture

In [2] two different scenarios of Geometric Monitoring corresponding to different network topologies are examined. The *decentralized scenario* refers to a topology where nodes are allowed to communicate with each other and a central node is absent. The *centralized scenario* models a star network topology, where a coordinator node communicating with all other nodes is existent.

Decentralized Scenario

The topology examined is that of a partially or fully connected mesh network where a coordinator node is absent and nodes are allowed to broadcast to the network or communicate with each other according to the links existent between them. Data stream update vectors arrive continuously at each of the monitoring nodes and nodes must always be synchronized, i.e. all nodes must be aware of the monitoring task's state at all times. An example is depicted in Figure 2.1.

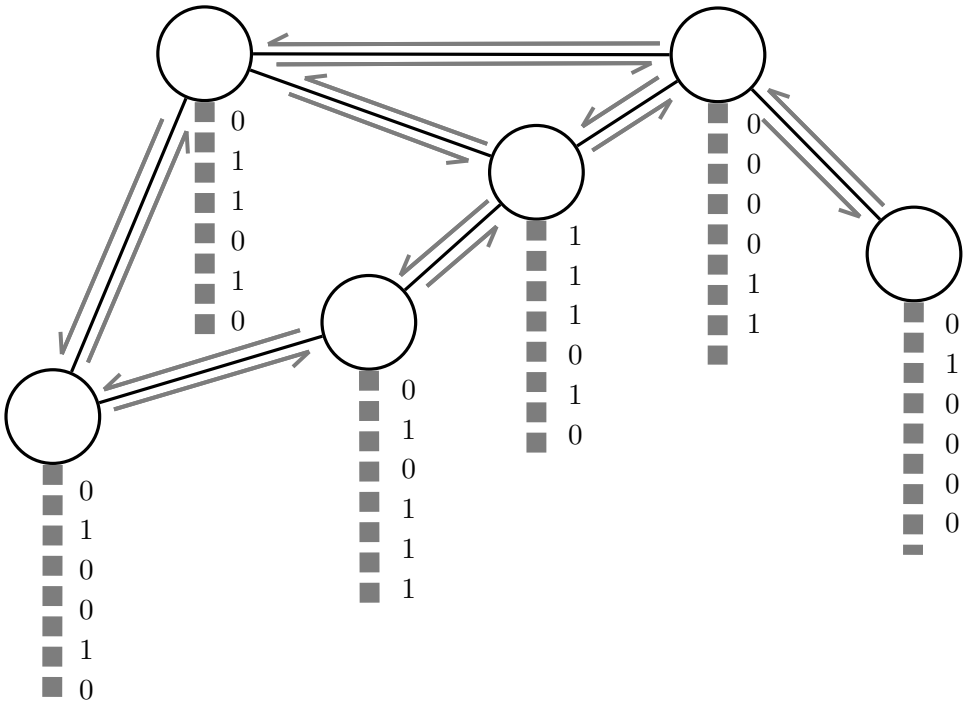


Figure 2.1: Network topology example of the decentralized scenario. Dashed lines represent data streams and half arrows represent message exchanges.

Centralized Scenario

The *centralized*, or *coordinator-based* scenario is built upon a star network topology, where all monitoring nodes communicate with a central node, the *coordinator node*. Nodes receive data

stream update vectors continuously, and must communicate their state information to the coordinator node when needed. The coordinator receives data stream updates as well, which can be modelled by an additional monitoring node responsible for the cooridinator node's data stream. Communication between monitoring nodes is not allowed, thus, only the communicator can, and must, be aware of the state of the monitoring task at all times. An example is depicted in Figure 2.2.

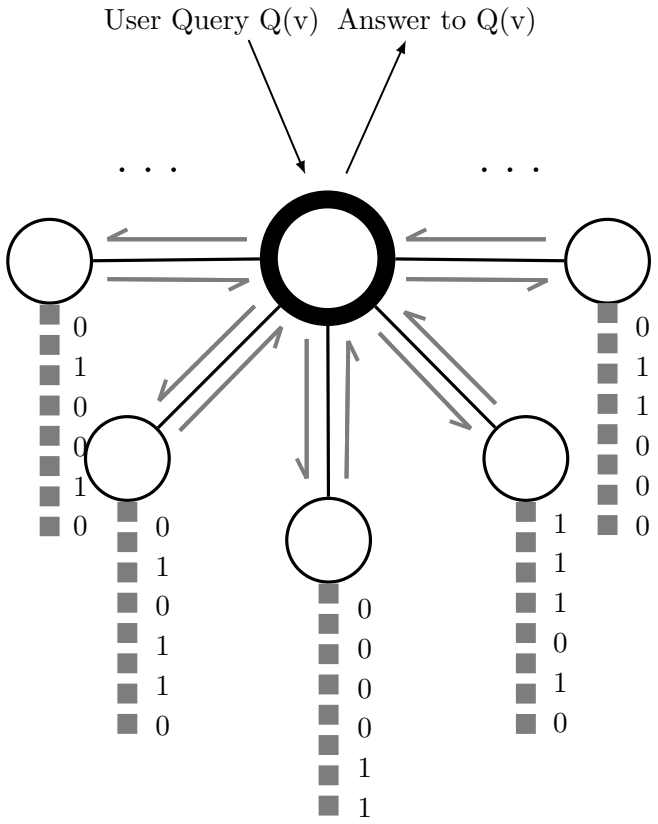


Figure 2.2: Network topology example of the centralized scenario.The bold node represents the coordinator node. Dashed lines represent data streams and half arrows represent message exchanges.

2.1.2 Computational Model

The main goal of the Geometric Monitoring method is to efficiently detect threshold crossings of an arbitrary function over distributed data streams. This is realized via vector projections of the data streams and convex local constraint assignments regarding said vectors at the nodes.

Let $f : \mathbb{R}^d \to \mathbb{R}$ be an arbitrary function, the *monitoring function*, whose value over the data streams needs to be monitored, so that if $f(\cdot) > T$ or $f(\cdot) < T$ an alarm is raised. For linear

functions this problem is trivial, so that by letting, for example, x_1 and x_2 be data stream values at different nodes and requiring $f(\frac{x_1+x_2}{2}) > 10$ to be monitored, it holds that $f(\frac{x_1+x_2}{2}) = \frac{f(x_1)+f(x_2)}{2}$, and the problem can be decomposed to local constraints $f(x_i) < 10, i = 1, 2$ at both nodes, i.e. a node remains silent until it violates its local constraint. Consider now the case of a non-linear function. By knowing the value of the function at the nodes nothing can be deduced about the function's value over the average of the monitoring streams and where it is positioned with respect to the threshold. Let $f(x) = 10x - x^2$, $x_1 = 0$ and $x_2 = 9$. Even thought $f(x_1) = 0 < 10$ and $f(x_2) = 9 < 10$, their average violates the specified threshold, $f(\frac{x_1+x_2}{2}) = f(4.5) = 24.75 > 10$.

In order to be able to effectively track non-linear functions, in the likes of the aforementioned example, a mapping of the streams to a vector space is taking place. Let $P = \{p_1, \ldots, p_n\}$ be the monitoring node set with weights $w_1, \ldots, w_n$, which can be either static or time varying. Their respective data streams $S = \{s_1, \ldots, s_n\}$ are represented by $\vec{v_1}(t), \ldots, \vec{v_n}(t)$, the d-dimensional *local statistics vectors* of the nodes at time t. The *global statistics vector* at time t is the weighted average of the local statistics vectors, as such:

$$\vec{v}(t) = \frac{\sum_{i=1}^{n} w_i \vec{v_i}(t)}{\sum_{i=1}^{n} w_i} \tag{2.1}$$

Infrequent communication between monitoring nodes, in the decentralized scenario, or between monitoring nodes and the coordinator, in the coordinator-based scheme, dictates the need to keep track of the value of the global statistics vector at the time the last global communication occurred, thus forming the *estimate vector*:

$$\vec{e}(t) = \frac{\sum_{i=1}^{n} w_i \vec{v_i}'}{\sum_{i=1}^{n} w_i} \tag{2.2}$$

,where $\vec{v_i}'$ is the last communicated statistics vector of node p_i.

At the monitoring nodes the difference between the current local statistics vector and the last communicated statistics vector is denoted by $\Delta\vec{v_i}(t) = \vec{v_i}(t) - \vec{v_i}', i = 1, \ldots, n$. The *drift vector* $\vec{u_i}(t), i = 1, \ldots, n$, also maintained at the monitoring nodes, represents the deviation of each node's data stream from the estimate vector and is defined differently in the two scenarios:

- In the **decentralized** setting the drift vector is regarded as the displacement of the local statistics vector from the estimate vector:

$$\vec{u_i}(t) = \vec{e}(t) + \Delta\vec{v_i}(t) \tag{2.3}$$

- In the **centralized** setting the monitoring nodes forward their state to the coordinator node, who has a global overview of the monitoring task at hand. This property allows the coordinator to counteract the effects a specific stream has on the partially observed monitoring task with an other, "opposite", stream belonging to a different monitoring node. This is taken care by the *balancing process* initiated every time a local violation occurs, which is responsible for computing and communicating the *slack vector* $\vec{\delta_i}$ to the nodes that contributed to the process, thus providing them with the necessary disposition of their drift vectors, as such:

$$\vec{u_i}(t) = \vec{e}(t) + \Delta\vec{v_i}(t) + \frac{\vec{\delta_i}}{w_i} \tag{2.4}$$

Balancing Process

The balancing process taking place in the **centralized scenario** is initiated by the coordinator node every time a threshold violation occurs, with the objective of resolving a possibly false alarm with minimal communication overhead. This task is executed by collecting a subset of monitoring nodes' data, the *balancing set* P', until the average of their drift vectors, the *balancing vector*, does not cause a threshold crossing. The balancing vector is formulated as follows:

$$\vec{b} = \frac{\sum_{p_i \in P'} w_i \vec{u_i}(t)}{\sum_{p_i \in P'} w_i} \tag{2.5}$$

After a successful balancing process has come to an end, $\Delta\vec{\delta_i}$ slack vector adjustments for all participants in the balancing set P' are computed and communicated to their respective sites, so that local drift vectors can be readjusted to reflect the balancing operation by computing $\vec{\delta_i} = \vec{\delta_i}' + \Delta\vec{\delta_i}$, where $\vec{\delta_i}'$ the previous slack vector (Equation 2.4). These adjustments are calculated as follows:

$$\Delta\vec{\delta_i} = w_i\vec{b} - w_i\vec{u_i}(t) \;\forall\; p_i \in P' \tag{2.6}$$

, where $\sum_{p_i \in P'} \Delta\vec{\delta_i} = \vec{0}$. Once the slack vector adjustments have been communicated to the respective monitoring nodes participating in P', their drift vectors are essentially set to the value of the newly computed balancing vector.

In case the balancing process proves unsuccessful all monitoring nodes are contained in the balancing set P' and a new estimate vector is computed with the data cumulated at the coordinator node. Subsequently, all drift vectors and slack vectors are set to $\vec{0}$.

2.1.3 Geometric Interpretation

The estimate vector, being the product of the system's previous global synchronization, is known to all monitoring nodes and denotes the last known position of the global statistics vector. That being said, the estimate vector is considered valid if it resides on the same side of the threshold as the unknown global statistics vector. In order to estimate the current position of the global statistics vector, since a mere observation of the monitoring function's value at each stream provides no information about its current location (as described in Section 2.1.2), it is vital that the task is decomposed into local constraints that will guarantee the timely detection of a violation of the estimate's vector validity.

The *convexity property* of the drift vectors, along with Theorem 1 [2], are sufficient in provide a framework for decomposing the monitoring task into local constraints at the nodes. Both the convexity property and the relevant theorem are repeated below for completeness.

The convexity property dictates that the weighted average of the drift vectors equal the global statistics vector, as such:

$$\vec{v}(t) = \frac{\sum_{i=1}^{n} w_i \vec{u_i}(t)}{\sum_{i=1}^{n} w_i} \tag{2.7}$$

The geometric interpretation of the property guarantees that the global statistics vector $\vec{v}$ is always contained in the convex hull defined by the drift vectors $\vec{u_i}, i = 1, \ldots, n$.

Theorem 1 (Sharfman et al. [2])**.** *Let $\vec{x}, \vec{y_1}, \ldots, \vec{y_n} \in \mathbb{R}^d$ be a set of vectors in $\mathbb{R}^d$. Let $Conv(\vec{x}, \vec{y_1}, \ldots, \vec{y_n})$ be the convex hull of $\vec{x}, \vec{y_1}, \ldots, \vec{y_n}$. Let $B(\vec{x}, \vec{y_i})$ be a ball centered at $\frac{\vec{x}+\vec{y_i}}{2}$ and with radius of $\|\frac{\vec{x}+\vec{y_i}}{2}\|_2$ i.e., $B(\vec{x}, \vec{y_i}) = \{\vec{z} \mid \|\vec{z} - \frac{\vec{x}+\vec{y_i}}{2}\|_2 \leq \|\frac{\vec{x}+\vec{y_i}}{2}\|_2\}$, then $Conv(vecx, \vec{y_1}, \ldots, \vec{y_n}) \subset B(\vec{x}, \vec{y_i})$.*

Essentially, Theorem 1 states that n d-dimensional spheres defined by $n+1$ vectors can effectively bound the convex hull defined by said vectors, as such: $Conv(\vec{x}, \vec{y_1}, \vec{y_2}, \ldots, \vec{y_n}) \subset \cup B(\vec{x}, \vec{y_i}), i =$

$1, \ldots, n$, which finds direct application to the distributed monitoring task if $\vec{x} = \vec{e}$ and $\vec{y_i} = \vec{u_i}, i = 1, \ldots, n$. An example is depicted in Figure 2.3 .

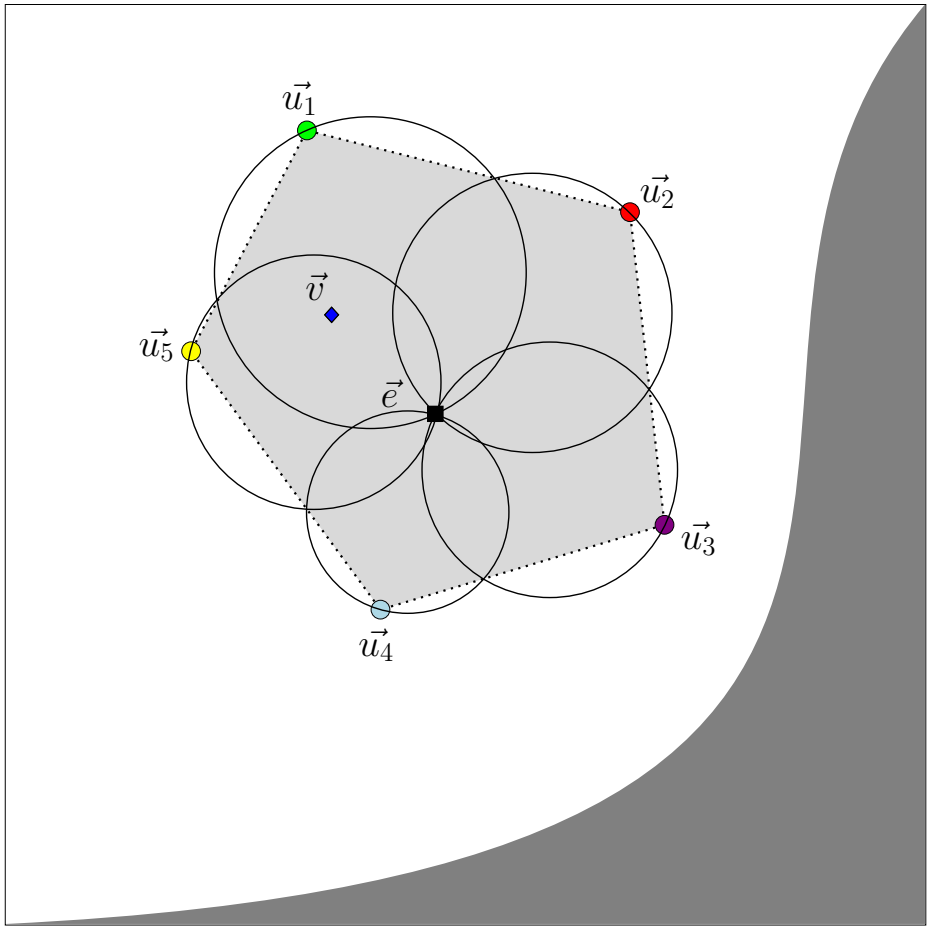


Figure 2.3: Example of a convex hull (light gray) defined by the drift vectors $\vec{u_i}, i = 1, 2, 3, 4, 5$. The hull is bounded by the spheres created from the estimate vector $\vec{e}$ and the drift vectors $\vec{u_i}, i = 1, 2, 3, 4, 5$. The global statistics vector $\vec{v}$ is guaranteed to be contained in the convex hull of the drift vectors.

Local Constraints

The decomposition of the threshold monitoring task to local constraints at the nodes, in which each node monitors its respective bounding sphere $B(\vec{e}, \vec{u_i}), i = 1, \ldots n$ for a possible threshold violation, induces a coloring upon the spheres. Let $V = \{\vec{x} | f(\vec{x} > T)\}$ be the set of vectors said to be *green*, and $\overline{V} = \{\vec{y} | f(\vec{y} < T)\}$ the *red* set of vectors, then the local constraint monitoring at the nodes is essentially a process of monitoring the monochromaticity of a node's bounding sphere $B(\vec{e}, \vec{u_i})$ i.e., all vectors in the bounding ball are of the same color. As long as this monochromaticity

is upheld for the whole of the node set, the convex hull defined by the drift vectors is monochromatic and, by the convexity property, the global statistics vector has not crossed the threshold. In case a single node signals a threshold crossing a *local violation* has occurred. If the local violation coincides with a threshold crossing of the global statistics vector, then a *global violation* has occurred.

2.1.4 Protocol

Two variants of a network's topological structure have been proposed for application of the Geometric Monitoring method, a decentralized scenario and a centralized, coordinator-based one (Section 2.1.1). The following paragraphs present the algorithms for each of these systems.

Decentralized Algorithm

The decentralized scenario of the geometric monitoring method, summarized in Algorithm 1, operates on the mesh network described in Section 2.1.1. Each node p_i keeps track of its drift vector $\vec{v_i}(t)$ and the previously communicated statistics vectors $\vec{v_j}'$ from all other nodes p_j, from which the estimate vector is locally computed. At the occurence of a local violation the violating node initiates a global system synchronization by broadcasting its local statistics vector along with its unique identifier, from which the estimate vector is globally updated so that monochromaticity checks are valid.

Algorithm 1: Decentralized algorithm

```
1  begin
2      foreach node p_i do                                   /* Node initialization */
3          Broadcast v_i(0);
4          v_i' = v_i(0);
5          Wait messages from all other nodes;
6          if messages from all vectors received then
7              e(t) = (Σ_{i=1}^{n} w_i v_i') / (Σ_{i=1}^{n} w_i);
8          end
9      end
10     foreach node p_i do                                   /* Main monitoring task */
11         foreach new s_i stream update v_i(t) do
12             Recalculate u_i(t) = e(t) + Δv_i(t);
13             if B(e, u_i(t)) is not monochromatic then
14                 Broadcast message < i, v_i(t) >;
15                 Set v_i' = v_i(t);
16             end
17             if new message < j, v_j(t) > received then
18                 Set v_j' = v_j(t);
19                 Recalculate e(t) = (Σ_{i=1}^{n} w_i v_i') / (Σ_{i=1}^{n} w_i);
20                 if B(e, u_i(t)) is not monochromatic then
21                     Broadcast message < i, v_i(t) >;
22                     Set v_i' = v_i(t);
23                 end
24             end
25         end
26     end
27 end
```

Centralized Algorithm

The centralized, coordinator-based geometric monitoring operation is summarized in Algorithms 2, and 3, where the execution sequence of the monitoring nodes and the execution sequence of the coordinator node are described, respectively. The topology is that of a star network, where

nodes are allowed to communicate exclusively with the coordinator node, as described in Section 2.1.1. The coordinator node is responsible for answering queries about the monitoring status i.e., has absolute knowledge about threshold violations, and handles the balancing process (Section 2.1.2). Local streams are tracked by the monitoring nodes on the basis of the last communicated estimate vector, and must inform the coordinator for any local threshold violation. The coordinator node can also monitor is respective data stream without any change in the described framework.

Algorithm 2: Centralized algorithm's monitoring node operation

```
1  begin
2      foreach node p_i do                                   /* Node initialization */
3          Send < INIT, v_i(0) > message to coordinator;
4          v_i' = v_i(0);
5          δ_i = 0;
6          Wait message from coordinator;
7          if < NEW-EST, e > message received then
8              Set e(t) = e;
9          end
10     end
11     foreach node p_i do                                   /* Main monitoring task */
12         foreach new s_i stream update v_i(t) do
13             Recalculate u_i(t) = e(t) + Δv_i(t) + δ_i / w_i;
14             if B(e, u_i(t)) is not monochromatic then
15                 Send < REP, v_i(t), u_i(t) > message to coordinator;
16                 Wait for < NEW-EST, · > or < ADJ-SLK, · > message from coordinator;
17             end
18             if new message < REQ > received then
19                 Send < REP, v_i(t), u_i(t) > message to coordinator;
20                 Wait for < NEW-EST, · > or < ADJ-SLK, · > message from coordinator;
21             end
22             if new < NEW-EST, e > message received then
23                 Set e(t) = e;
24                 v_i' = v_i(t);
25                 δ_i = 0;
26             end
27             if new < ADJ-SLK, Δδ_i > message received then
28                 δ_i = δ_i + Δδ_i;
29             end
30         end
31     end
32 end
```

Algorithm 3: Centralized algorithm's coordinator node operation

begin
  Wait for $< INIT, \cdot >$ messages from all monitoring nodes; `/* Initialization */`
  $\vec{e}(0) = \frac{\sum_{i=1}^{n} w_i \vec{v_i}(0)}{\sum_{i=1}^{n} w_i}$;
  if *new* $< REP, \vec{v_i}(t), \vec{u_i}(t) >$ *message received* **then** `/* Monitoring operation */`
    $P' = P' \cup \{< i, \vec{v_i}(t), \vec{u_i}(t) >\}$;
    `Balance(`P'`);`
  end
end
Function `Balance(`P'`)` `/* Balancing Process */`
  $\vec{b} = \frac{\sum_{p_i \in P'} w_i \vec{u_i}(t)}{\sum_{p_i \in P'} w_i}$;
  if $B(\vec{e}, \vec{b})$ *is* not *monochromatic* **then**
    if $P - P' \neq \emptyset$ **then**
      Send $< REQ >$ message to random node in $P - P'$ set;
    else
      $\vec{e}(t) = \frac{\sum_{i=1}^{n} w_i \vec{v_i}(t)}{\sum_{i=1}^{n} w_i}$;
      Send $< NEW\text{-}EST, \vec{e}(t) >$ message to all nodes;
      return;
    end
  else
    foreach $p_i \in P'$ **do**
      $\Delta\vec{\delta_i} = w_i \vec{b} - w_i \vec{u_i}(t)$;
      Send $< ADJ\text{-}SLK, \Delta\vec{\delta_i} >$ message to node p_i;
      return;
    end
  end
end

2.2 Multi-objective Optimization

Multi-objective optimization, also known as multi-objective programming, vector optimization and Pareto optimization, belongs to the field of *decision making* and focuses on mathematical optimization problems. As it is evident by the term, multiple, possibly conflicting, objectives exist and are required to be simultaneously optimized. Such problems arise in a multitude of fields, from

engineering to finance and molecular studies.

One example application originating from the field of aeronautics is the optimization of objectives such as speed, travel range, fuel consumption, safety and aircraft building costs by taking into account decision variables in the likes of engine trust, number of engines, wall thickness, wing area and luggage capacity. Attempts at optimizing such problems usually lead to a plethora of optimal solutions, where trade-offs must be made regarding the decision variables.

Optimal solutions, where none of the objective functions can be improved without the simultaneous degradation of other objective functions' values, are called *non-dominated*, or *Pareto optimal solutions*. A formalization of a multi-objective optimization framework is stated in Equation 2.8.

Let vector of m objectives $F(x) = [F_1(x), F_2(x), \ldots, F_m(x)]$:

$$\begin{aligned} &\min_{x \in \mathbb{R}^n} F(x) \\ &\text{s.t. } l \leq x \leq u \\ &\qquad G_i = 0, i = 1, \ldots, k_e \\ &\qquad G_j \leq 0, j = k_e + 1, \ldots, k \end{aligned} \tag{2.8}$$

, where $x \in \mathbb{R}^n$ is the decision variable vector, l and u denote the respective lower and upper bounds of x, G_i are the equality constraints and G_j are the inequality constraints the solution must uphold. The decision variable vector is said to exist into the *decision variable space*, and the objective vector lies in the *objective space*. A mapping of the feasible set under F forms the *attained set* $C = \{y \in \mathbb{R}^m | y = F(x), x \in \mathbb{R}^n\}$. A graphical representation of the Pareto optimal solutions creates the *Pareto front*, *Pareto curve*, or *Pareto surface*, as shown in Figure 2.4.

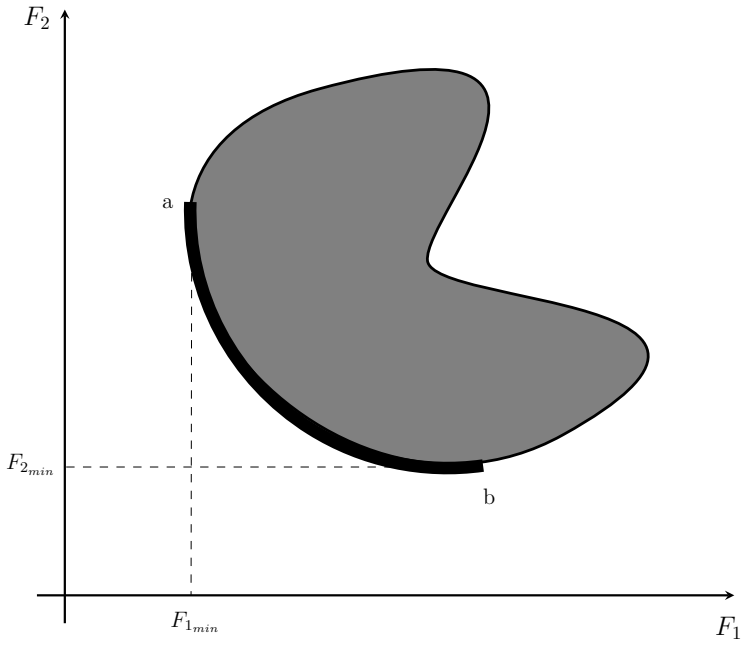


Figure 2.4: Example of the objective space of a multi-objective optimization problem with two objective functions. The feasible region is shaded with gray, and the respective Pareto front is denoted with bold. Points a, and b mark the optimal points for each of the two depicted objective functions, F_1 and F_2 respectively.

Finding the Pareto optimal solution to such problems is generally *NP-hard* in complexity. Thus, various approximation methods exist that either lead to the optimal solution, if this is available, or provide a solution set approximation in the case of non-available or partially available Pareto fronts. These methods originate from different viewpoints of the multi-objective optimization problem and can be divided into numerical and evolutionary optimization algorithms, with our focus being targeted towards the former.

2.2.1 Non-linear Constrained Optimization Problems

Solutions to optimization problems where the objective functions are generally non-linear and both equality and inequality constraints exist are usually provided by iterative methods similar to *line search* for single objective optimization problems. At each iteration t an appropriate direction d_t and a successive point x_{t+1} is chosen given the current position x_t. Following this paradigm a sequence of points $\{x_t\}_{t=1}^{\infty}$ and directions $\{d_t\}_{t=1}^{\infty}$ are produced until the maximum iteration limit is reached or convergence has been achieved. A generic primal descent algorithm is shown in

Algorithm 4.

Algorithm 4: Generic primal descent

```
1 begin
2     Choose initial point x_0 ∈ X and set t = 0 ;        /* Initialization */
3     while Termination condition not satisfied do        /* Search */
4         t = t + 1;
5         Determine search direction d_t;
6         Determine step length s_t, so that f(x_t + s_t d_t) < f(x_t);
7         Update ;
8     end
9 end
```

Feasible Directions

The method of *feasible directions* for constrained function minimization attempts to iteratively converge to an optimal point on the basis of Algorithm 4 by employing *usable feasible directions.*

A search direction d_t is termed as *usable feasible direction* if it satisfies two properties:

1. a small disposition towards direction d_t does not violate any constraint i.e.,

$$d_t^T \nabla G(x_t) \leq 0$$

2. a move towards d_t reduces the objective functions value i.e.,

$$d_t^T \nabla F(x_t) < 0$$

.

In case the feasible region D is convex the line connecting the optimal point, $x^\star$, with any other arbitrary point $x \in D$ lies completely inside the convex region and is, thus, reachable via the feasible directions method.

SQP

Following the framework of non-linear constrained optimization algorithms the *sequential quadratic programming* almost feasible point methods attempt to solve problems by quadratically

approximating non-linear objective functions subject to linearly approximated equality and inequality constraints by decomposing the original problem to a sequence of quadratic programming subproblems. Such methods do not always produce feasible points during iterations, but ultimately feasibility is enforced.

Given the general case of the multi-objective optimization problem in Equation 2.8 a *Lagrangian function* is formed:

$$\mathcal{L}(x, \lambda) = F(x) + \sum_{i=1}^{k} \lambda_i G_i(x) \tag{2.9}$$

, with λ being *Lagrangian multipliers*. Based on the newly created function a decomposition to quadratic programming subproblems is taking place, where non-linear constraints are linearized and inequality constraints substitute the bound constraints found in Equation 2.8, as such:

$$\begin{aligned} \min_{d \in \mathbb{R}^n} \ & \frac{1}{2} d^T H_t d + \nabla F(x_t)^T d \\ & \nabla G_i(x_t)^T d + G_i(x_t) = 0, i = 1, \ldots, k_e \\ & \nabla G_i(x_t)^T d + G_i(x_t) \leq 0, i = k_e + 1, \ldots k \end{aligned} \tag{2.10}$$

, with H_z being a Hessian matrix approximation at iteration t and d being the search direction. Subsequently, by obtaining a step length s_t through a line search method the following iteration point is computed, as stated in Algorithm 4.

2.3 Savitzky-Golay Filtering

The *Savitzky-Golay filter* [4] is a digital, low-pass smoothing filter following the paradigm of *moving window averaging* i.e.,

$$g_i = \sum_{n=-n_L}^{n_R} c_n f_{i+n}$$

,where the underlying function $f(\cdot)$, with $f_i = f(x_i)$ denoting the value of the function at data point x_i, is approximated over a window of size $n_L + n_R + 1$ by a higher order polynomial so that coefficients c_n retain higher moment information.

Assume equidistant data points and let the polynomial of order M:

$$y_i(x) = a_0 + a_1\frac{x - x_i}{\Delta x} + a_2(\frac{x - x_i}{\Delta x})^2 + \cdots + a_M(\frac{x - x_i}{\Delta x})^M$$

Firstly a *least squares fit* of the polynomial is taking place over the span of the window:

$$\sum_{j=i-n_L}^{i+n_R} (y_i(x_j) - f_j)^2 = \min$$

Subsequently the value of g_i is set to the resulting value of the fitted point x_i, and this process proceeds iteratively for all data points.

While a seemingly burdensome process, by considering that the least squares fitting requires just a single linear matrix inversion and that the coefficients a_i of the fitted polynomial are linear in the data values, the computation can be notably simplified to a pre-computation of the smoothing coefficients and a subsequent convolution.

Following a matrix notation, we define the matrix $\mathbf{J}$ containing the $n_L + n_R + 1$ points corresponding to each order of the polynomial:

$$\mathbf{J} = \begin{bmatrix} 1 & -n_L & \dots & (-n_L)^M \\ \vdots & \vdots & & \vdots \\ 1 & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 1 & n_R & \dots & n_R^M \end{bmatrix} \in \mathbb{R}^{(n_L+n_R+1)\times(M+1)}$$

The vector $\mathbf{a}$ containing the polynomial coefficients:

$$\mathbf{a} = \begin{bmatrix} a_M \\ \vdots \\ a_1 \\ a_0 \end{bmatrix} \in \mathbb{R}^{M+1}$$

The vector $\mathbf{f}$ of the $n_L + n_R + 1$ original data points:

$$\mathbf{f} = \begin{bmatrix} f_{i-n_L} \\ \vdots \\ f_i \\ \vdots \\ f_{i+n_R} \end{bmatrix} \in \mathbb{R}^{n_L+n_R+1}$$

Thus, the least squares fitting can be written as :

$$\|\mathbf{Ja} - \mathbf{f}\|_2 = \min$$

By solving the resulting normal equations:

$$\mathbf{a} = (\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T\mathbf{f}$$

the polynomial coefficients can be computed. Finally, the *convolution coefficients* are contained in:

$$\mathbf{C} = (\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T$$

,and the smoothed signal can be easily computed as such:

$$g_i = (\mathbf{C}e_{M+1})^T\mathbf{f}$$

, with e_{M+1} being the $(M+1)$st unit vector.

From the above it can be seen that the value of the resulting signal at the center point is obtained from a single set of coefficients, while the remaining sets are able to produce the desired derivatives of the original signal. By incorporating a set of data over a window of length n_L+n_R+1 for the computation of a single point it is assumed that the redundancy present in distant data aids at increasing the signal to noise ratio.

2.4 Matching in Graphs

Let $G = (V, E)$ be a graph with V being the vertex set and E being the set of edges connecting said vertices. In graph theory a *matching* forms a subset of edges $M \subseteq E$, so that no two edges share a common vertex, with a *perfect matching* covering the whole vertex set of the graph. Subsequently, a *maximum matching* is defined as the matching M with the largest possible number of edges, and a *maximum weight matching* is the matching M that maximizes the sum of edge weights.

Maximum Weight Matching, the Primal-Dual method

The *Primal-Dual method* for maximum weight matching in graphs [5] is based of the duality found in Linear Programming problems.

Specifically, let a Linear Programming optimization problem (Equation 2.12), its *Dual Linear program* (Equation 2.13) is formulated so that its variables, the *dual variables*, model the constraints of the original problem, while its constraints represent the *primal variables* of the original problem. This allows optimization of the primal problem's value by tightening its bounds, as computed by the dual program. By optimizing the value and retaining feasibility of the dual program the -not necessarily feasible- primal problem approaches feasibility. Finally, due to equivalence between the primal program and its dual, the algorithm terminates with both optimal primal and dual solutions. This relationship is depicted in Equation 2.11.

$$\begin{aligned} \text{Constraints in Primal} &\iff \text{Variables in Dual} \\ \text{Constraints in Dual} &\iff \text{Variables in Primal} \end{aligned} \tag{2.11}$$

The general linear program formulation, along with its dual, are shown in Equations 2.12 and 2.13, respectively:

$$\begin{aligned} \max \quad & \mathbf{c}^T\mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{aligned} \tag{2.12}$$

$$\begin{aligned} \min \quad & \mathbf{b}^T\mathbf{y} \\ \text{s.t.} \quad & \mathbf{A}^T\mathbf{y} \geq \mathbf{c} \\ & \mathbf{y} \geq 0 \end{aligned} \tag{2.13}$$

,where $\mathbf{A} \in \mathbb{R}^{M \times N}$, $\mathbf{b} \in \mathbb{R}^{M}$, $\mathbf{c} \in \mathbb{R}^{N}$, $\mathbf{x} \in \mathbb{R}^{N}$ are the primal variables, and $\mathbf{y} \in \mathbb{R}^{M}$ are the respective dual variables.

Following this paradigm the maximum weight matching problem can be formulated as Primal-Dual linear programming problem. By defining a positive weight function on the vertices $y : V \to \mathbb{R}^{+}$, a *weighted vertex cover* is a subset $C \subseteq V$ such that $\forall e = (u, v) \in E, u, v \in C : y_u + y_v \geq w_{u,v}$. Additionally, let a matching M and $x_{u,v} = 1$ iff edge $e_{u,v} = (u, v) \in M$. The resulting linear programming pair is depicted in Equations 2.14 and 2.15, the former being the primal program and the latter being its respective dual program.

$$\begin{aligned} \max \quad & \sum_{(u,v)\in E} x_{u,v} w_{u,v} & & \\ \text{s.t.} \quad & \sum_{u \in e : e \in E} x_e \leq 1 & , u \in V & \qquad (2.14) \\ & x_{u,v} \geq 0 & , (u, v) \in E & \end{aligned}$$

$$\begin{aligned} \min \quad & \sum_{u \in V} y_u & & \\ \text{s.t.} \quad & y_u + y_v \geq w_{u,v} & , (u, v) \in E & \qquad (2.15) \\ & y_u \geq 0 & , u \in V & \end{aligned}$$

Chapter 3

Related Work

A great deal of prior work exists on threshold monitoring and monitoring of distributed sets of data streams, mostly focusing on applications where the monitored function is well defined and linear. In [6] the sum of a distributed set of variables is monitored for threshold crossings, with [7] proposing the addition of local constraints to reduce communication overhead. Continuous tracking and approximate answering of specific aggregation operations (sum, averaging and minimum) over a coordinator-based scenario is explored in [8]. Additionally, [9,10] provide methods for estimating simple functions over distributed data streams. k-largest aggregate value monitoring is described in [11], where local constraint enforcement and efficient resolution of false constraint violations is presented.

Elevation of the restriction of monitoring function linearity happens at [2], where a geometric method for threshold monitoring of arbitrary functions over coordinator-based and mesh-like network topologies is described. In [12] approximate answers to complex aggregate queries are provided by a coordinator node, with the distributed nodes retaining synopses of the monitored data and communicating them when local constraints are being violated i.e., significant divergence of local data from the previously communicated data has been observed. Furthermore, prediction mechanisms are employed in order to reduce the communication burden. By reducing the approximate query answering to local threshold crossing monitoring at the distributed nodes, [3] succeeds in unifying the continuous monitoring task with the geometric threshold monitoring method of [2].

Safe Zones are introduced in [13] as an extension of the geometric monitoring method of [2], where an arbitrary function is geometrically monitored by employing optimal local constraints at the nodes, that are fitted to each node's data distribution. In order to reduce the computational burden of optimal local constraint formation a hierarchical node clustering scheme is implemented that allows recursive computation of the problem at hand. In [14] the Safe Zones and the bounding

balls of the geometric monitoring method are proven to be fundamentally the same. Following that claim, [15] explores ellipsoidal bounds as a way to minimize the volume of bounding regions and reduce the communication overhead induced by false alarms. Additionally, communication of temporal data, along with first and second moments of the nodes' data distributions, as well as a method for decoupling the estimate vector from it's use as the reference vector for bounding region construction is proposed. Constraints tailored to fit data distributions at the nodes are also explored in [16], where simple and efficiently computable shapes, as well as a hierarchical clustering of the monitoring nodes into disjoint sets that maximize the probability of resolution of false alarms, are proposed. Finally, [17] offers a generalization of the geometric monitoring scheme by incorporating a variety of prediction models based on velocity and acceleration of the vector representations of the data streams.

Part II

PROBLEM DEFINITION AND IMPLEMENTATION

Chapter 4

Problem Statement

The geometric monitoring method of [2] provides a rigid framework of projecting streams into an Euclidean, multi-dimensional space in order to effetively monitor threshold crossings of arbitrary functions. Subsequent work attemts to provide solutions to various drawbacks of the original method by optimizing local constraints at the distributed nodes, exploring node clustering schemes that will provide deterministic and efficient methods for optimal constraint computation and node selection during the balancing process, and employing forecasting models to predict and limit Local Violations that do not result to a threshold crossing of the aggregate stream.

While these attempts succeed at reducing the communication overhead of the geometric monitoring method, a scalability problem still persists regarding the dimensionality of the data streams and the monitoring node population size. This thesis explores the limits of the balancing process itself, and how optimal positioning of drift vectors into space, a significant aspect of the method, not touched upon in prior work, as well as appropriate node selection for inclusion into the balancing set, can improve monitoring performance and reduce the communication cost, while taking into acount termporal stream properties, such as velocity and acceleration.

Chapter 5

Implementation

This chapter provides a detailed description of the implemented system. In Section 5.1, the Geometric Monitoring method implementation is described, along with the necessary simplifying assumptions to aid experimentation. Following that, in Section 5.2 an algorithm for node matching is proposed, inspired by the violation recovery method found in [16]. In Section 5.3, the heuristic based balancing method for local violation resolution is presented, along with the necessary data stream tracking scheme. Finally, the main implementation challenges are discussed.

5.1 Geometric Monitoring Implementation

The initial Geometric Monitoring method [2], which is described in detail in Section 2.1, provides two algorithms for threshold monitoring of distributed data streams. These algorithms operate on different network structures and implement a somewhat different handling of threshold violations.

The decentralized algorithm operates on a coordinator-less environment, where nodes are allowed to communicate with each other, whereas the coordinator-based algorithm has a Star network topology, where the coordinator node is the central node (the *hub*) and the Monitoring nodes reside on the edges of the network. The algorithm operating on the decentralized setting does not provide a balancing process for local violation resolution. On the other hand, the coordinator based algorithm implements a violation resolution operation every time a local violation occurs, which aims to minimize the communication overhead induced by false violation reports.

Our focus is centered towards a simplified **coordinator-based algorithm** (Algorithm **??**), described in Section 2.1, as it provides a framework for the heuristic balancing process, as well as the node matching operation presented in detail in Sections 5.3 and 5.2 respectively.

To aid method formulation and experimentation, the following simplifying assumptions have been made regarding the coordinator-based algorithm:

- Communication between nodes is considered instantaneous. There is no delay when passing messages through the network. The problem of message handling in a real-world Geometric Monitoring method implementation, where message delays are induced by the underlying network, has been studied in detail in [18].

- Communication between nodes is considered loss-less and reliable. In case network reliability can not be guaranteed appropriate methods should be considered.

- The system operates in an iterative fashion, as described in Algorithm 5. This simplification of the real-time distributed monitoring process to an iterative process provides a more manageable setting for experimentation without distorting the results of the proposed methods, which can be applied directly to the original real-time distributed setting.

- The system pauses at each violation, until the violation is resolved. During violation resolution Monitoring nodes do not receive updates from their respective data streams.

- The Coordinator node does not participate in the monitoring operation. The Coordinator node does not receive updates from a data stream, it only receives messages from the Monitoring nodes in case of threshold violation. This assumption can easily be elevated by considering an additional monitoring node responsible for handling the coordinator's data stream monitoring operation.

Algorithm 5: Iterative network operation

```
Data: monitoringNodes: a list of Monitoring nodes, coordinator: the Coordinator node
1  begin
2      initialization;
3      repeat
4          foreach node ∈ monitoringNodes do
5              node.DataVectorUpdate();
6              node.ComputeDriftVector();
7          end
8          foreach node ∈ monitoringNodes do
9              node.CheckForViolation();
10             if localViolation then
11                 node.Report();
12                 coordinator.Balance();
13             end
14         end
15     until globalViolation;
16 end
```

5.2 Distance Based Node Matching

The balancing method of the coordinator-based algorithm, as described in Section 2.1 [2, 15], aims at resolving local violations that do not result in a global violation (*false alarms*) by balancing the violating node's drift vector with the respective vectors of *randomly* chosen nodes. Consider the violating node n_i with weight $w_i = 1$, so that the bounding ball $B(\vec{e}(t), \vec{u_i}(t))$ is not monochromatic, and the randomly requested node n_j with weight $w_j = 1$, so that the newly formed bounding ball is $B(\vec{e}(t), \frac{\vec{u_i}(t)+\vec{u_j}(t)}{2})$, where $\vec{e}(t)$ the estimate vector at time t and $\vec{u_i}(t)$, $\vec{u_j}(t)$ the drift vectors of nodes n_i, n_j at time t, respectively. If the resulting bounding ball is monochromatic the violation is resolved, otherwise another node is *randomly* requested for balancing.

As observed in [16,19], the original balancing method's node choosing scheme can be inefficient, so a more efficient and deterministic approach should be adopted. Optimal pairing of nodes and the construction of a hierarchical structure (Figure 5.1) reduces the communication overhead of false alarms, with the vast majority of violation resolutions requiring only the assigned node pair to be successful. The criterion by which nodes are paired attempts to maximize the probability

of a successful balance by maximizing *"the percentage of pairs of data vectors from both nodes whose sum is in the Minkowski sum of the nodes' safe-zones"* [16], or, in this case, whose resulting bounding ball is monochromatic.

Here, the same node pairing scheme is followed, but with a different, distance based, criterion for grouping nodes into disjoint pairs and creating the hierarchical structure depicted in Figure 5.1. The method proceeds as follows (Algorithm 6):

1. Monitoring nodes are visualized as the nodes of a complete graph $G = (V, E)$, where $V = \{n_1, n_2, ..., n_k\}$ vertex set consists of the initial Monitoring nodes (*"Type-1 nodes"*) and $E = \{(n_i, n_j)\ \forall i, j \in [1, ..., k], i \neq j\}$ edge set contains an edge for every pair of vertices.

2. Weights are assigned to all edges E. The weight of each edge is defined as the cumulative distance of the value of the monitoring function on the mean of each pair of data vectors from the value of the monitoring function on the *global* mean of all Monitoring nodes' data vectors, plus the cumulative distance of each pair of data vectors:

$$w_{i,j} = \sum_{t=t_0}^{t_{end}} [(f(\vec{v}_{global}(t)) - f(\frac{\vec{v_i}(t) + \vec{v_j}(t)}{2})) + (|\vec{v_i}(t) - \vec{v_j}(t)|)] \tag{5.1}$$

   , where $\vec{v_i}(t)$ the data update of node n_i at time t, $\vec{v}_{global}(t)$ the global mean of all Monitoring nodes at time t and $f(\cdot)$ the monitoring function.

3. Maximum weighted matching is performed on the resulting graph via the *primal-dual* method implemented in the *networkx* Python library [20], so that nodes are partitioned into disjointed sets M_i, $|M_i| = 2\ \forall i \in [1, ..., \frac{k}{2}]$.

4. Each set $M_i, i \in [1, ..., \frac{k}{2}]$ is considered a single node, so that a new complete graph $G' = (V', E')$ is created, where $V' = \{M_1, ..., M_{\frac{k}{2}}\}$ (*"Type-2 nodes"*) the new vertex set and $E' = \{(M_i, M_j)\ \forall i, j \in [1, ..., \frac{k}{2}]\}$ the new edge set. Weights are assigned to the new edges and the process repeats until the resulting graph contains only a single vertex (*"Type-k node"*), which incorporates all the initial Monitoring nodes.

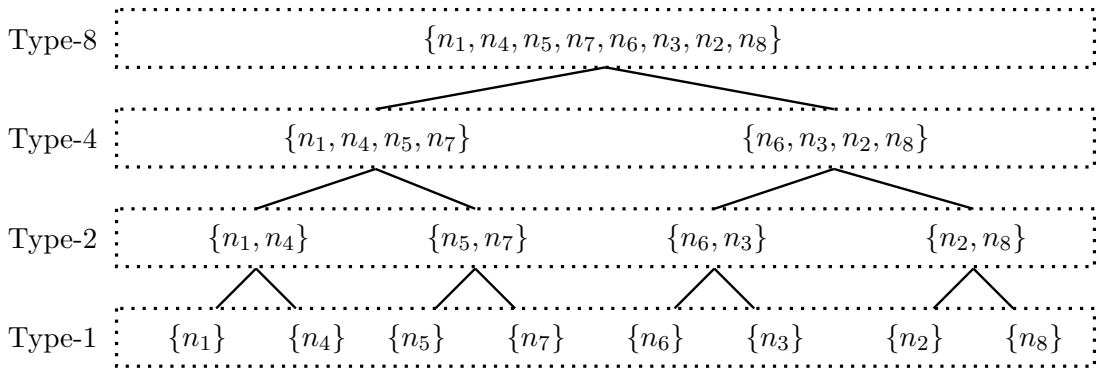


Figure 5.1: Hierarchical pairing scheme example for node set $\{n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8\}$.

5. Vertices not matched with any other vertex during the matching process are ignored in future iterations. During the balancing process such unmatched vertices are handled by the traditional random selection balancing algorithm found in [2](also, Section 5.1).

Algorithm 6: Recursively create Monitoring node pairs and hierarchy

Function `DistancePairer(`*nodes,i*`)`

Data: $nodes = [(n_1, [\vec{v_1}(t_0), ..., \vec{v_1}(t_{end})]), ..., (n_k, [\vec{v_k}(t_0), ..., \vec{v_k}(t_{end})])]$: list of nodes with their respective data vectors,i: pair type, initial=1

Result: *nodeHierarchy*: dictionary of *Type-k* pairs

if $length(nodes) = 1$ **then** `// recursion stopping condition`

  return *nodeHierarchy*;

end

$g = CreateCompleteGraph(nodes)$; `// complete graph with` *nodes* `as vertices`

foreach $(n_i, n_j) \in g.Edges()$ **do** `// assign weights to edges`

  $w_{i,j} = \sum_{t=t_0}^{t_{end}} [(f(\vec{v}_{global}(t)) - f(\frac{\vec{v_i}(t)+\vec{v_j}(t)}{2})) + (|\vec{v_i}(t) - \vec{v_j}(t)|)]$;

  $g.edge(n_i, n_j).weight = w_{i,j}$;

end

$nodeHierarchy$(Type-i) $= g.maximalWeightMatching()$; `// node pairs of` *Type-i*

`DistancePairer(`$nodeHierarchy($*Type-i*$), i * 2$`);`

end

The incentive behind the distance based node pairing scheme comes from the need to track the global data vector as closely as possible, with only a subset of the total node population's data vectors at each balancing attempt. By considering the distance of the mean of a pair of data vectors from the global data vector (distance d_1 in Figure 5.3) the *"quality"* and *"accuracy"* of the tracking

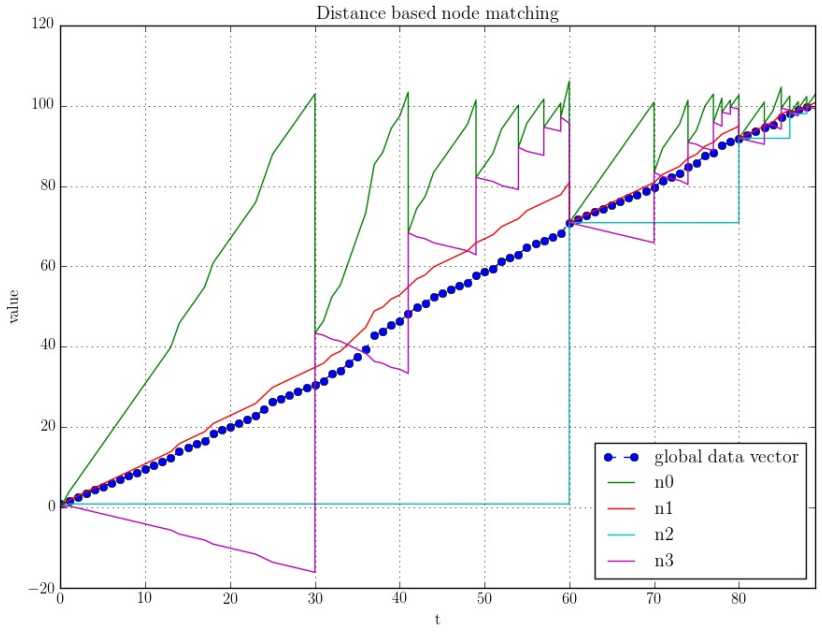


Figure 5.2: The drift vectors during Geometric Monitoring operation until a Global Violation. Distance based node matching is used on 4 nodes ($\{n_0, n_1, n_2, n_3\}$), with 1-dimensional data vectors, threshold $T = 100$ and $f(x) = x$ as the monitoring function. The *Type-2* node pairs are $\{n_0, n_3\}$ and $\{n_1, n_2\}$.

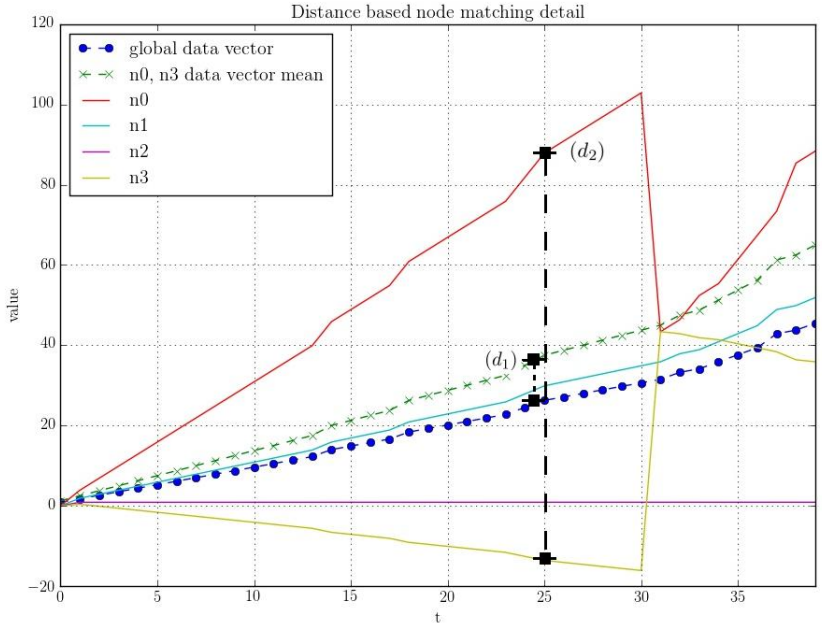


Figure 5.3: Detailed depiction of the Geometric Monitoring operation of Figure 5.2. Distance based node matching operating on 4 nodes ($\{n_0, n_1, n_2, n_3\}$), with 1-dimensional data vectors, threshold $T = 100$ and $f(x) = x$ as the monitoring function. Distance d_1 denotes the distance of the data vector mean of the paired nodes n_0 and n_3 from the global mean (*global data vector*) at $t = 25$, whereas distance d_2 denotes the in-between distance of data vectors $\vec{v_0}(t)$ and $\vec{v_3}(t)$ of the node pair at time $t = 25$ (before a Local Violation occurs, where $\vec{e} = 0$ and $\vec{u_i}(t) = \vec{v_i}(t)\ \forall\ i \in [0, 1, 2, 3], t < 30$). Both distances are taking part in the edge weighting process, according to Equation 5.1.

ability of each pair is evaluated. Additionally, by taking into account the in-between distance of data vectors of each node pair (distance d_2 in Figure 5.3), pairs from the limits of the data vector velocity spectrum that manage to *"cancel each other out"* more effectively are encouraged.

5.3 Heuristic Balancing

The balancing method incorporated into the *coordinator based* algorithm of the Geometric Monitoring method [2] (Section 2.1) attempts to minimize the communication overhead of local violations by computing the, so called, *balancing vector*. The *balancing vector* is defined as the weighted mean of the drift vectors of the nodes contained in the balancing set, and, in case of a successful balance, it is guaranteed that $B(\vec{e}, \vec{b})$ is monochromatic. Consequently, by setting the drift vectors of the nodes in the balancing set to be equal to the balance vector, all local constraints are fulfilled and the convexity property of the drift vectors is satisfied.

While this method partially succeeds in reducing the communication burden of false alarms either by requesting only a subset of the total node set each time a Local Violation occurs or by setting the drift vectors to a safe point (represented by the balance vector), major drawbacks can be noted regarding vector positioning and bounding ball construction. Updated vector assignment as a result of the "optimization" procedure does not take into account the idiosyncrasies of the monitoring function and the admissible region it produces. Additionally, all nodes taking part in the balancing process are handled identically, without taking advantage of the differences in the behavior of each node.

Previous work proposed selecting an optimal reference vector, instead of the estimate vector for bounding ball construction, along with shape customization of the local constraints at the nodes according to the node's needs [15]. Local constraint customization served as the basis for the now popular *Safe-Zone* framework [13, 16], which diverges from the traditional bounding sphere setting, while maintaining the same fundamental idea of distance computation of a point from a set of support vectors [14], preserving the essence of the admissible region and retaining the balancing process of the coordinator based scenario.

This thesis proposes a novel heuristic approach for optimal positioning of drift vectors, which takes into account both the temporal behavior of each node's data stream, as well as the peculiar-

ities of the monitoring function over said data streams. Aim of the heuristic optimization is the maximization of the estimated time until the following Local Violation occurs, which, expressed as an optimization formula, receives the following form:

$$\max \min \frac{(T - x_i) - accel_i(t_{lv}) * t^2}{vel_i(t_{lv})}, \forall n_i \in P' \tag{5.2}$$

where:

t : the variable to optimize

T : monitoring threshold

x_i : the maximum value of the monitoring function $f(\cdot)$ over the bounding ball $B(\vec{e}(t_{lv}), \vec{u_i}(t_{lv}))$, where t_{lv} is the time a Local Violation occurred and i the index of node n_i

$vel_i(t_{lv})$: the estimated velocity of the maximum value of the monitoring function $f(\cdot)$ when applied to the bounding ball created by the data stream update of node n_i and the estimate vector $\vec{e}$ at time t_{lv}

$accel_i(t_{lv})$: the estimated acceleration of the maximum value of the monitoring function $f(\cdot)$ when applied to the bounding ball created by the data stream update of node n_i and the estimate vector $\vec{e}$ at time t_{lv}

t_{lv} : time of Local Violation occurrence

P' : the balancing set

The Equation 5.2 originates from elementary kinematic equations, as such:
Assume a moving object i at point x_i, with acceleration a_i and current velocity v_i. Let v_f be the object's final velocity when it reaches a threshold point T at time t, from which it deviates by $d = T - x_i$. Let current time be $t = 0$.
Distance (or *Displacement*) in terms of velocity and acceleration is described by:

$$d = v_i t + at^2 \tag{5.3}$$

For which it holds:

$$
\begin{aligned}
d &= v_i t + a_i t^2 \leftrightarrow \\
T - x_i &= v_i t + a_i t^2 \leftrightarrow \\
t &= \frac{(T - x_i) - a_i t^2}{v_i}
\end{aligned}
$$

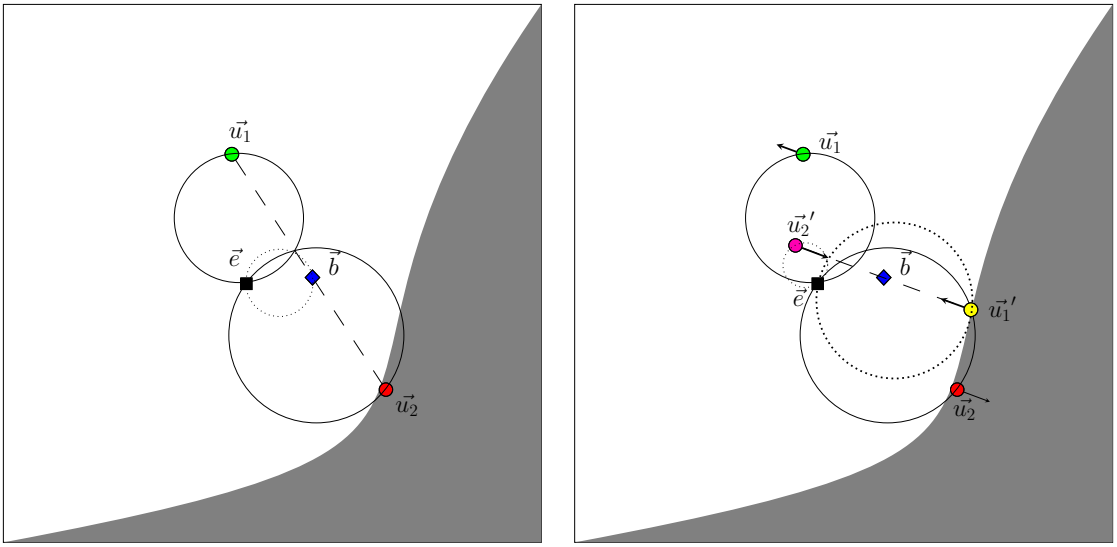
Thus, t is the expected time the moving object reaches the threshold point T.

The newly defined heuristic optimization formula (5.2) aims to maximize the time until the next Local Violation concerning any of the nodes belonging in the balancing set. By taking into account the maximum value of the monitoring function $f(\cdot)$ inside the bounding ball created by each data stream update and the estimate vector, and by computing acceleration and velocity measures of this value over time, an approximate mapping of the data stream space to the one dimensional space of the arbitrary monitoring function is achieved. This permits the computation of the optimal positions the balanced drift vectors should take in order to maximize the time they reach the monitoring threshold, as depicted in Figure 5.4b.

5.3.1 Implementation of the Heuristic Balancing

In order transform the heuristic optimization formula (5.2) into an applicable setting, *multi-objective optimization* (Section 2.2) is used. The optimization function is now defined as such:

$$
\begin{aligned}
&\min -z \\
&\text{s.t.} \quad z \leq g(h(\vec{e}, \vec{u_0}), vel_0, accel_0, T) \\
&\qquad z \leq g(h(\vec{e}, \vec{u_1}), vel_1, accel_1, T) \\
&\qquad \vdots \\
&\qquad z \leq g(h(\vec{e}, \vec{u_n}), vel_n, accel_n, T) \\
&\qquad \vec{b} = \frac{1}{\sum_{i=0}^{n} w_i} \sum_{i=0}^{n} (w_i * \vec{u_i}) \qquad\qquad ,\forall n_i \in P'
\end{aligned}
\tag{5.4}
$$



(a) The classic balancing method. As long as $B(\vec{e}, \vec{b})$ is monochromatic (i.e. within the Admissible region), balance is successful and the updated drift vectors are set to $\vec{u_1}' = \vec{u_2}' = \vec{b}$.

(b) The heuristic balancing method. Arrows depict the velocities of each drift vector. After a successful balance is achieved ($B(\vec{e}, \vec{b})$ is monochromatic), the optimal points in which the updated drift vectors $(\vec{u_1}', \vec{u_2}')$ should be positioned are computed by maximizing the estimated time until the next Local Violation, based on the current drift vector positions and the estimated velocities. Balance vector $\vec{b}$ remains unchanged.

Figure 5.4: Balancing methods

where:

$g : \mathbb{R}^4 \to \mathbb{R}$, the heuristic optimization function as defined in Equation 5.2

$h : \mathbb{R}^d \to \mathbb{R}$, the function computing the maximum value of the monitoring function $f(\cdot)$ in $B(\vec{e}, \vec{u_i})$, which is an optimization problem by itself

d : the data vector dimensionality

T : the monitoring threshold

$\vec{u_i}$: the drift vector of node n_i

w_i : the weight of node n_i

vel_i : the velocity of the maximum value of the monitoring function when applied to the ball defined by node's n_i drift vector $\vec{u_i}$ and the estimate vector $\vec{e}$

$accel_i$: the acceleration of the maximum value of the monitoring function when applied to the ball defined by node's n_i drift vector $\vec{u_i}$ and the estimate vector $\vec{e}$

$\vec{b}$: the balancing vector

Solution to the above optimization problem (5.4) is given by a *Sequential Least Squares Programming (SLSQP)* solver, which implements *sequential quadratic programming* (described briefly in Section 2.2.1) by using the *Han-Powell quasi-Newton* method with *BFGS* update at each iteration for the Hessian matrix approximation, and an *L1-test function* for computing the step length. The solver is implemented by the *pyOpt* Python optimization library [21]. The problem is decomposed and formulated using an additional helping parameter z in order to avoid non-differentiable functions (such as min and max) and to aid computation by the solver.

In the heuristic optimization problem defined previously (5.4) the nested optimization of detecting the maximum value of an arbitrary monitoring function inside the bounding ball $B(\vec{e}, \vec{u_i})$ is existent. This optimization problem is formed as follows:

$$\max \quad f \tag{5.5}$$

$$\text{s.t.} \quad \sum_{i=1}^{d} (x_i - c_i)^2 = r^2 \tag{5.6}$$

where:

f : the monitoring function $f(\cdot)$

x_i : element i of d-dimensional vector $\vec{x}$

c_i : element i of d-dimensional vector $\vec{c}$, which represents the center of the sphere

r : the radius of the sphere

d : the space dimensionality

Eq. 5.6 : a $(d+1)$ dimensional sphere in $\mathbb{R}^d$

The optimization problem of detecting the maximum value of a function inside a sphere (5.5) is solved using *Constrained Function Minimization (CONMIN)*, which implements the method of feasible directions, as described in Section 2.2.1 and implemented by the *pyOpt* Python optimization library [22].

The resulting heuristic balancing algorithmic implementation is summarized in the following Algorithm:

Algorithm 7: Heuristic Balancing

```
1  Function RepMessageReceived(< n_i,v_i,u_i,vel_i,accel_i >)
2      add n_i to balancing set P';
3      Balance();
4  end
5  Function Balance(P')
6      if length(P') = 1 then
7          RequestNode();     // request node based on respective gathering scheme
8      end
9      b⃗ = Σ_{P'} (w_i * u_i⃗) / w_i;
10     if B(e⃗, b⃗) is monochromatic then
11         /* heuristic optimization procedure,                                  */
12         /* returns the optimal drift vector positions in set O               */
13         O = DriftVectorOptimizationProblem();
14         foreach n_i ∈ P' do
15             Δδ_i = w_i * u_i⃗' − w_i * u_i⃗; // u_i⃗' denotes the optimal drift vector position
16             Send(< ADJSLK, n_i, Δδ_i >);
17         end
18     end
19 end
```

5.3.2 Smoothing, Velocity and Acceleration Estimation via Savitzky-Golay

The heuristic balancing method proposed previously (Section 5.3) requires an efficient estimation of the velocity and the acceleration of the output of the monitoring function over the maximum value of the bounding ball. Additionally, a smoothing operation over the data stream series would be beneficial, in order to grasp the trend (increasing or decreasing) of the data stream without letting noisy updates and extreme fluctuations misguide the optimization operation.

The *Savitzky-Golay smoothing filter* [4] (Section 2.3) is ideal in the heuristic Geometric Monitoring setting, for it smooths and derivates the signal without much additional computational burden, allowing it to be applied directly at the Monitoring Nodes' data streams. By assuming equidistant data points the precomputation of convolution coefficients becomes trivial when specifying the window size, the window center, the order of the polynomial and the desired derivative. Following that, the precomputed coefficients are applied to the desired signal by a simple convo-

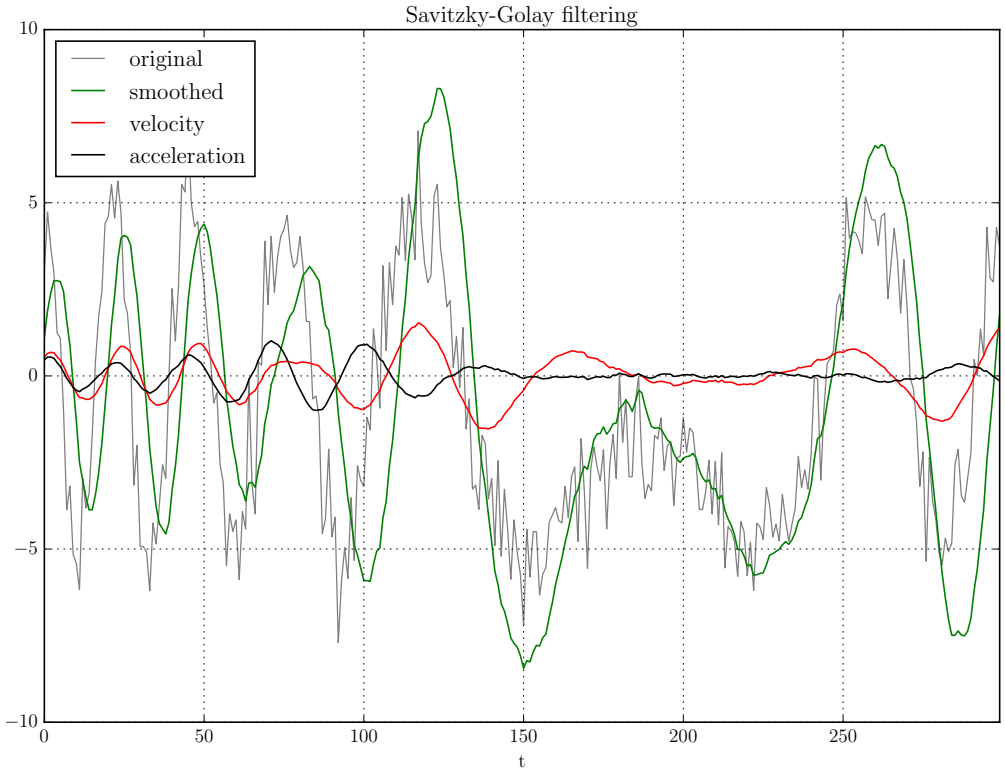


Figure 5.5: Savitzky-Golay filtering of a signal with added Gaussian noise. The smoothing window is 50 points in length, centered at the far end, as in the real-time smoothing applied to the Geometric Monitoring setting. The polynomial order is 2 for the smoothed signal, 3 for the velocity estimation and 5 for the acceleration estimation of the original signal.

lution, which is both fast and, if required, on-line. The application of the filter on a noisy signal, along with the velocity and the acceleration computation of this signal, is shown in Figure 5.5.

5.4 Implementation Challenges

The proposed methods and algorithms incur some implementation challenges, which, on the greater part, can be managed.

Regarding the *distance based node matching* presented in Section 5.2:

- In order to extract the optimal node pairs training data must be available. This situation can be handled in two ways. One way is to initiate execution of the Geometric Monitoring task using the original method of randomly requesting nodes during balancing, until the necessary amount of data to train the model has been cumulated. Then the model can be trained and

the operation can be switched to the distance based node matching scheme. A more appropriate solution could be to incrementally update the node pairs using the data provided by message passing in the standard Geometric Monitoring execution, or by occasionally polling the monitoring nodes during low network activity until a satisfiable amount of data has been gathered.

Regarding the *heuristic balancing* method presented in Section 5.3:

- The bi-level multi-objective optimization incorporated into the method can become computationally expensive when dealing with a large balancing set or with highly dimensional data streams. Attention must be paid to the selected solvers responsible for the optimization task, for some solvers can be more effective than others in different settings and different monitoring function applications. Additionally, some solvers provide customization parameters, such as *tolerance* and *iteration count*, among others, that greatly influence the execution time of the optimization routine, as well as the precision of the results.

- The *Savitzky-Golay smoothing filter*, responsible for smoothing and differentiating the signals representing the maximum value of the monitoring function over the bounding spheres, is directly affected by the selected window length and the polynomial order. That being the case, care must be taken to select appropriate values that effectively track the general trends without compromising detail important to the optimization routine.

Part III

RESULTS AND CONCLUSIONS

Chapter 6

Experiments

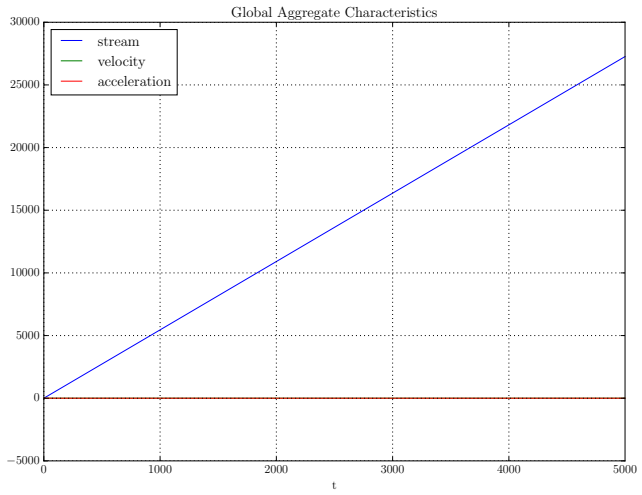
In order to evaluate our proposed algorithms, the heuristic balancing method (GM), the distance based node clustering scheme ($DIST$), as well as the combination of those (HDM), we compare them with the geometric monitoring method [2] (GM), and the hierarchical clustering method of [16] ($DISTR$). Datasets originate from both synthetic and real-world settings in order to explore the performance, scalability and applicability of our methods in terms of reduction in communication.

Firstly, Section 6.1 contains a detailed description of the datasets and the monitoring functions used for evaluation purposes. Following that, Section 6.2 presents the experimental results along with the necessary commentation.

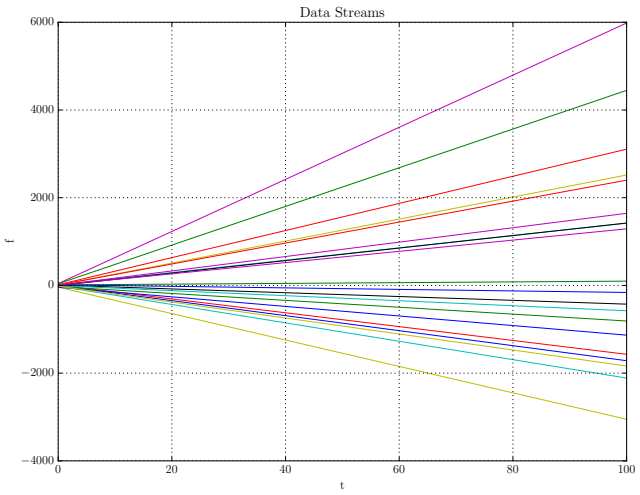
6.1 Data, Setup and Monitoring Functions

6.1.1 Synthetic Datasets

Synthetic datasets have been incorporated into the evaluation process of the proposed algorithms in order to provide a controllable environment under which the behavior of our methods can be analyzed. Data streams are created by firstly sampling data stream velocity distribution means by a user specified normal distribution and fixing the standard deviation of each stream. Afterwards, an initial velocity is sampled from each stream's assigned distribution and a λ value is chosen, which controls the rate of change of the streams' velocity. Stream update $v_i(t_k)$ of node n_i at time t_k is generated by sampling a new velocity u_{k+1} from the velocity distribution assigned to node n_i and updating the streams' value by: $v_i(t_{k+1}) = v_i(t_k) + (1 - \lambda)u_k + \lambda u_{k+1}$. Noisy versions of the generated streams are the product of additive Gaussian noise.

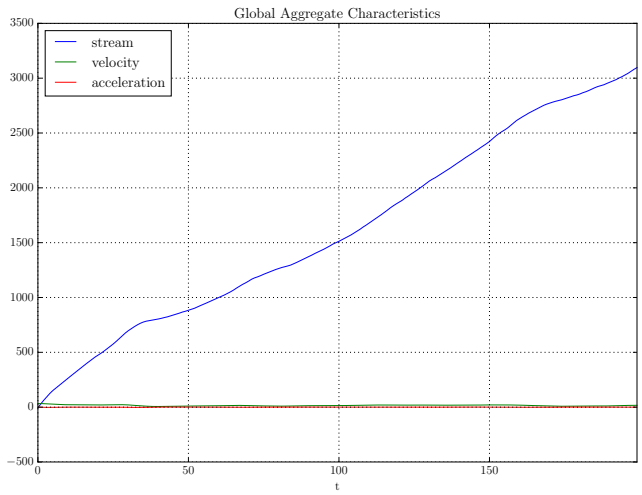


(a) LIN global statistics stream of 20 streams

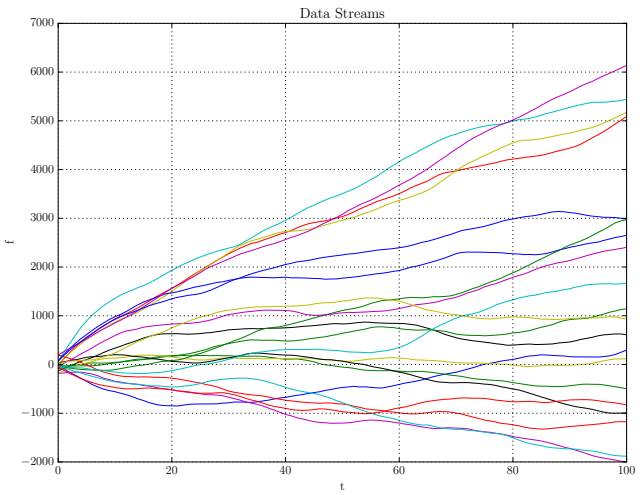


(b) LIN local statistics streams of 20 nodes

Figure 6.1: Linear data stream examples (LIN)



(a) INT global statistics stream of 20 streams

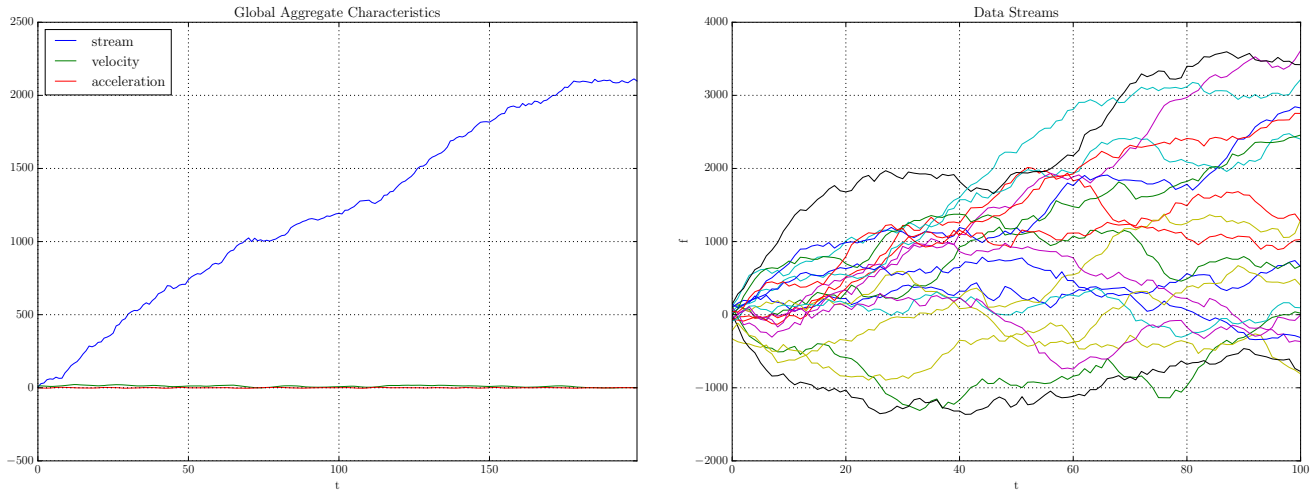


(b) INT local statistics streams of 20 nodes

Figure 6.2: Interweaving data stream examples (INT)

Linear streams (LIN) are generated by setting the parent distribution to $\mathcal{N}(10, 20)$, the standard deviation of each stream's velocity distribution to $\sigma = 10$ and the lambda value to $\lambda = 0$. An example of the resulting one dimensional streams corresponding to 20 nodes, along with the resulting global statistics stream, is illustrated in Figure 6.1.

Interweaving streams(INT) are produced from the same parent distribution $\mathcal{N}(10, 20)$ by selecting $\sigma = 50$ for the standard deviation of each stream's velocity distribution and $\lambda = 0.1$ as the



(a) NOISE global statistics stream of 20 streams (b) NOISE local statistics streams of 20 nodes

Figure 6.3: Interweaving data stream examples (NOISE)

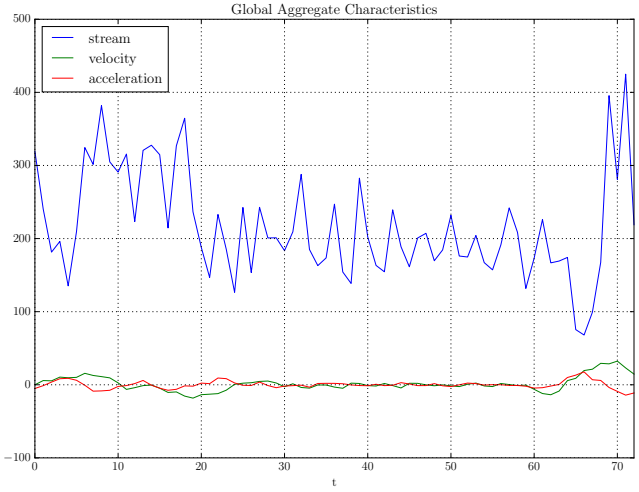
velocity update parameter. An instance of one dimensional interweaving streams corresponding to 20 nodes, along with the resulting global statistics stream, is shown in Figure 6.2.

Noisy streams (*NOISE*) are the result of the previously used parent distribution with $\sigma = 100$ as the standard deviation of each stream's velocity, $\lambda = 0.15$ the velocity update parameter, and $\mathcal{N}(0, 30)$ the distribution of the additive Gaussian noise. Such one dimensional streams corresponding to 20 monitoring nodes, along with the resulting global statistics stream, are depicted in Figure 6.3.

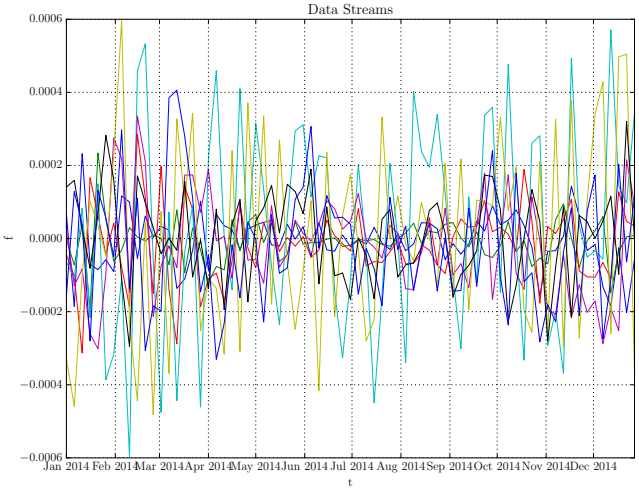
When necessary, a partitioning of the dataset into training and testing sets is executed by uniformly sampling 20% of the initial dataset as the training set, with the rest being the testing dataset.

6.1.2 Air Quality Database

The real world dataset consists of measurements of air pollutants, as measured during the year 2014, provided by the "European Environmental Agency - AQ e-Reporting" database [23]. Data streams correspond to hourly measurements of air pollutants NO_2 and NO, in micro-grams per cubic meter, averaged over a window of five days for a whole year. Monitoring nodes are picked at random from available air quality measurement stations across Austria. Notable characteristics of this dataset are the difference in behavior and shape between data streams taken from different

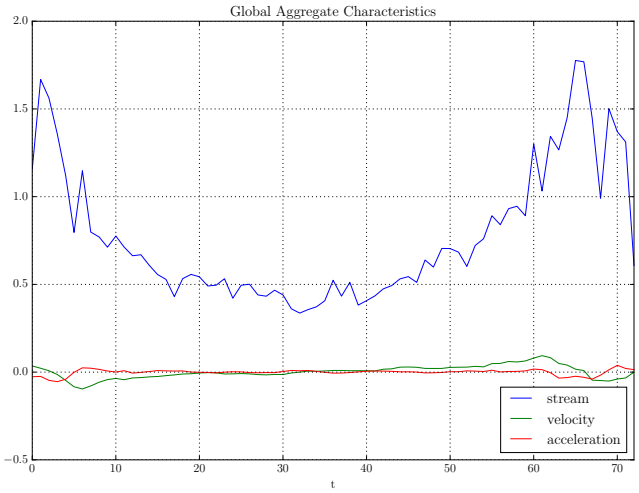


(a) Global statistics stream of 8 nodes monitoring the variance of NO_2 air pollutant.

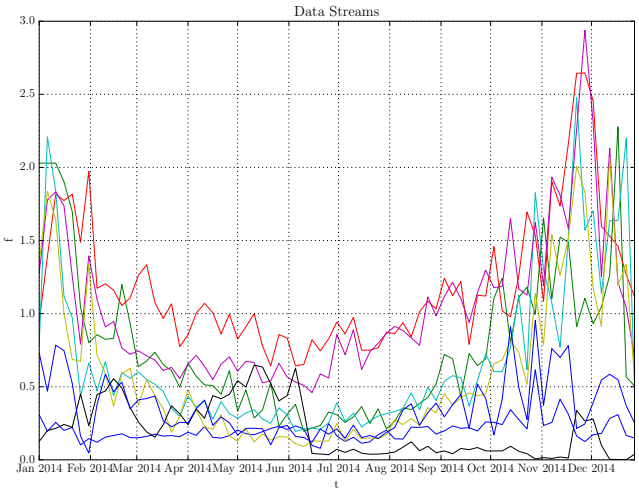


(b) Local statistics streams of 8 nodes monitoring the variance of NO_2 air pollutant.

Figure 6.4: Streams of 8 nodes monitoring the variance of NO_2 air pollutant.



(a) Global statistics stream of 8 nodes monitoring the ratio NO/NO_2.



(b) Local statistics streams of 8 nodes monitoring the ratio NO/NO_2.

Figure 6.5: Streams of 8 nodes monitoring the ratio NO/NO_2.

stations, and between different air pollutant measurements. These lead to irregularities and great variance between measurements at different time points and locations. Figures 6.4 and 6.5 illustrate the global and local statistics streams of 8 nodes that monitor the variance of NO_2 pollutant, as well as the ratio of NO to NO_2.

Where applicable the training dataset is regarded as the first month of the year.

6.1.3 Monitoring Functions

The monitoring functions used during the experiments were carefully chosen in order to illustrate, as accurately as possible, the properties and behavior of each examined method. Specifically:

- Experiments using the one dimensional synthetic datasets (*LIN, INT, NOISE*) monitor the function $f(x) = x$. This simple functions allows us to clearly examine the behavior of the implemented methods over artificial streams with specific characteristics regarding linearity and noise, without affecting the results.

- Experiments that incorporate multi-dimensional synthetic datasets (*LIN, INT, NOISE*) monitor a multi-variable quadratic function $f(x, y, z, k, \dots) = (x - y + z - k + \dots)^2 + x + y + z + k + \dots$, with variables $x, y, z, k, \dots$ corresponding to different stream dimensions i.e., a quadratic function with d variables is monitored over d-dimensional streams. Quadratic functions are of grave importance to numerous real-world applications (e.g., a Gaussian distribution is expressed via an exponent of a quadratic function).

- Experiments performed on real-world data streams of air pollutants monitor the variance of NO_2 and the ratio of NO to NO_2. Both functions operate on two dimensional data. Let m_{NO_2,t_i} and m_{NO,t_i} be measurements of air pollutants NO_2 and NO at t_i, respectively. The former function operates on data updates $v_{t_i} = \begin{pmatrix} m_{NO_2,t_i} \\ (m_{NO_2,t_i})^2 \end{pmatrix}$ and the latter on data updates $v_{t_i} = \begin{pmatrix} m_{NO,t_i} \\ m_{NO_2,t_i} \end{pmatrix}$.

6.2 Experimental Results

The following experiments are performed in order to gain an insight on the behavior of the methods proposed in this thesis and how these compare to methods presented in prior work, focusing on the communication overhead induced by each method until a Global Violation takes place.

In Subsection 6.2.1 our distance based node matching algorithm (Section 5.2) is compared with two methods of violation resolution found in related work. Subsection 6.2.2 compares the balancing algorithm of the seminar work on geometric monitoring with our proposed, heuristic, method. Subsequently,in Subsection 6.2.3 our HDM method is compared with the GM method while exploring the impact different tuning parameters induce on the performance of our algorithm.

Finally, our algorithm HDM is put to test using datasets from a real-world domain, with GM operating as the baseline method, in Subsection 6.2.4.

6.2.1 Node Matching Algorithms

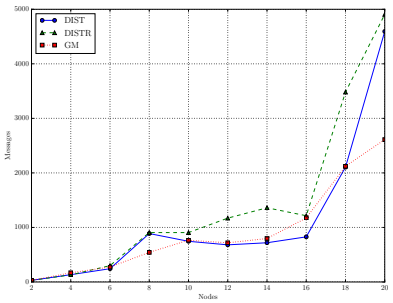
The seminar geometric monitoring method [2] dictates that random nodes are requested in order to perform a violation resolution via the balancing method (RAND). Subsequent work [16] examines the partitioning of nodes into disjoint pairs, so that the probability of a violation resolution is maximized by maximizing the percentage of data vectors that result to a successful balancing operation (DISTR), either by iterating over all data vectors pairs between all nodes and evaluating the constraint, or by employing the pdfs of the data streams. Our proposed method depends only on simple Euclidean distance computations between data stream vectors, not being bound on the monitoring function and its possible irregularities. All three methods are being compared alongside the original geometric monitoring balancing method GM.

Figure 6.6 provides information about the communication overhead each method incurs, and how this evolves as a function of node count. Method DIST proves to be comparable with the RAND method for node selection, both surpassing the DISTR method in terms of communication reduction. Specifically, RAND and DIST methods illustrate an equal of better performance than the DISTR method over the whole range of nodes, for all three datasets *LIN, INT* and *NOISE*. Algorithms RAND and DISTR mostly induce the same communication burden, with some exceptions located towards the larger end of the node spectrum, where for 16 nodes DIST method always surpasses the RAND method in terms of communication reduction, and for 20 nodes the RAND method induces less message exchange than the rest of the algorithms. This is due to the random selection DIST and DISTR algorithms perform in case pefect node pairs cannot be created i.e., the node number is not a power of 2.

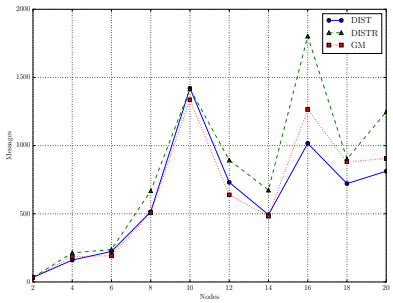
A graphical illustration of the required nodes for a successful violation resolution is provided in Figure 6.7. Following the trend of the aforementioned experiment, RAND and DIST methods achieve a violation resolution by employing far less nodes than the DISTR algorithm. Roughly 70 to 80 percent of all Local Violations are successfully resolved by 2 nodes by employing the RAND and DIST methods. The DISTR method is able to resolve false alarms by using only 2 nodes with

a score of 60 percent or less of total Local Violations, with the rest requiring more than 2 nodes to successfully perform the balancing process.

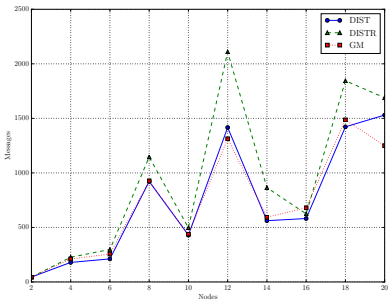
These results can be explained by viewing how each methods handles node pairing, in Figure 6.8. It is evident that the DIST method attempts to group together data streams that are more likely to "balance each other out" by taking into account the in-between distance, as well as the distance of the average of the two streams from the global data stream. On the other hand, the DISTR method assists node pairings whose average will not induce a Local Violation, while allowing streams that are more likely to violate the threshold be grouped together. Subsequently, when a node from the latter pair signals a threshold crossing, its pair will not be able to successfully resolve the violation, thus requesting additional nodes from a higher level of the hierarchical clustering.



(a) Communication costs of methods RAND,DISTR and DIST for the *LIN* dataset.



(b) Communication costs of methods RAND,DISTR and DIST for the *INT* dataset.

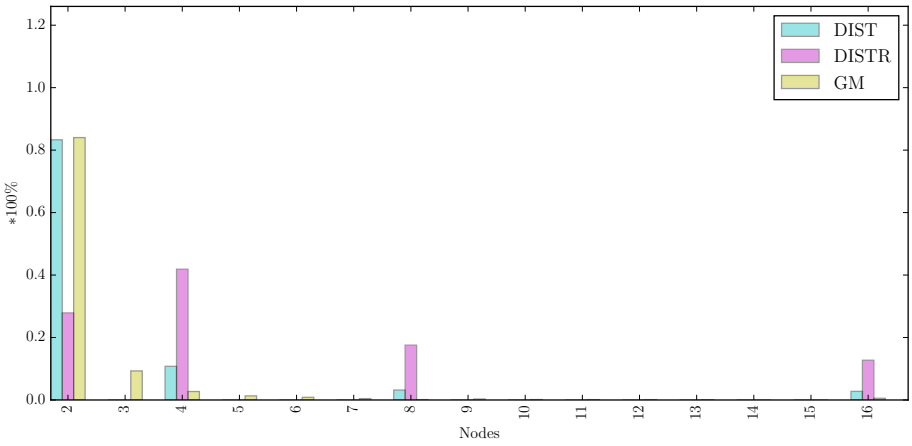


(c) Communication costs of methods RAND,DISTR and DIST for the *NOISE* dataset.

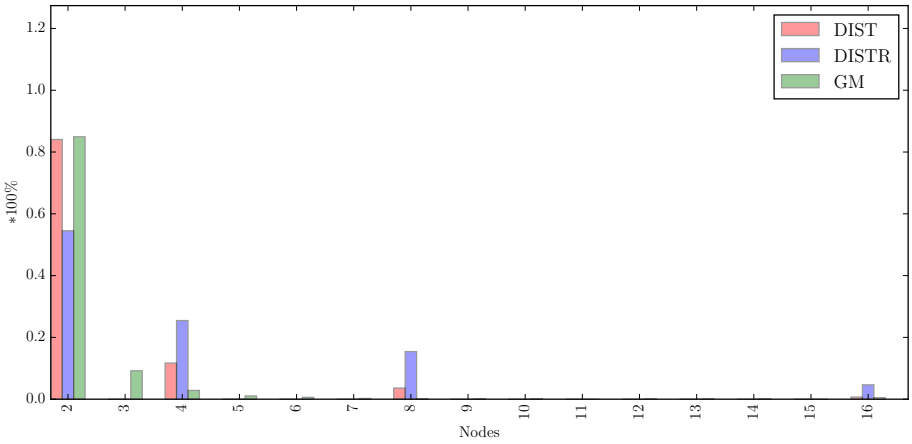
Figure 6.6: Comparison of RAND, DISTR and DIST methods in terms of communication cost in messages over the range of 20 nodes.

6.2.2 Balancing methods

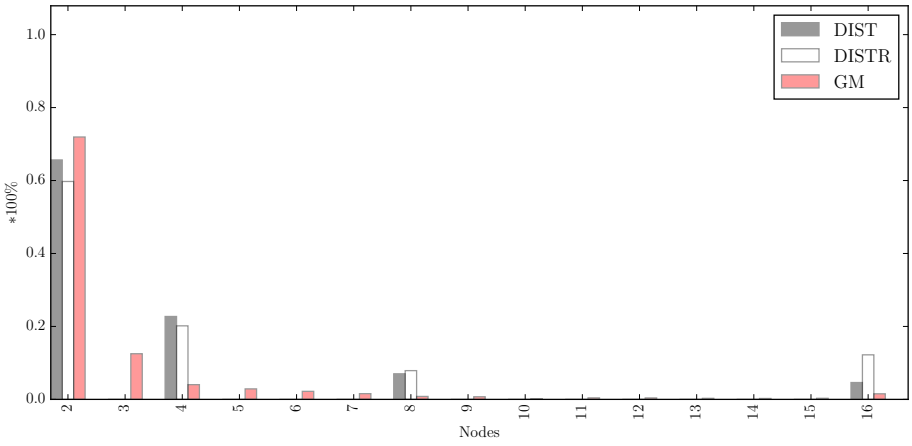
During the balancing process of the original geometric monitoring method [2] (GM) the computed drift vectors of the nodes are set to the value of the balancing vector that successfully resolves the reported Local Violation. In contrast, the heuristic balancing method proposed in this thesis (HM) attempts to optimize the positioning of the new drift vectors by taking into account the properties of each stream. Figure 6.9 illustrates the communication cost induced by the balancing methods over a range of 2 to 20 nodes for the datasets *LIN, INT* and *NOISE*, when applied along-



(a) Number of nodes participating in violation resolutions as a fraction of total Local Violations, for the *LIN* dataset.

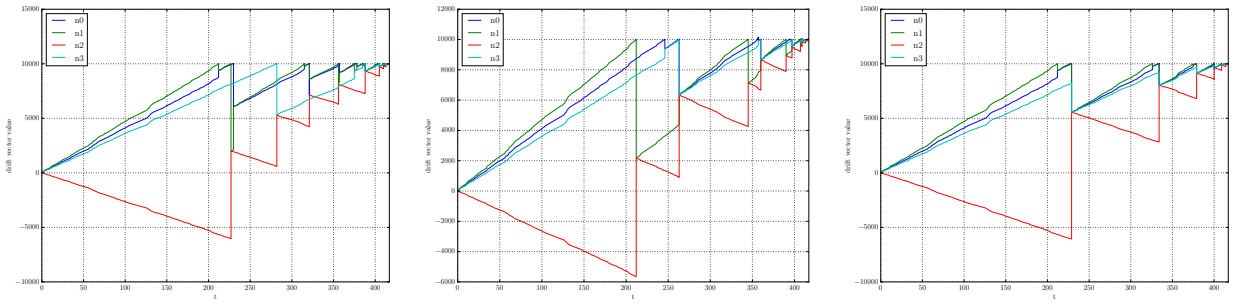


(b) Number of nodes participating in violation resolutions as a fraction of total Local Violations, for the *INT* dataset.



(c) Number of nodes participating in violation resolutions as a fraction of total Local Violations, for the *NOISE* dataset.

Figure 6.7: Number of nodes required to successfully resolve a violation, as a fraction of total Local Violations, for methods RAND, DISTR and DIST.



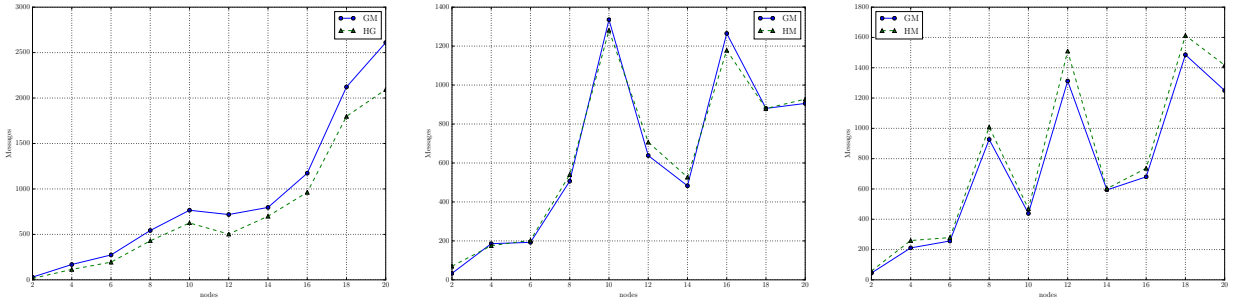
(a) Drift vectors of 4 nodes as a result of the RAND method.

(b) Drift vectors of 4 nodes as a result of the DIST method.

(c) Drift vectors of 4 nodes as a result of the DISTR method.

Figure 6.8: Drift vectors of 4 nodes as a result of application of RAND, DIST and DISTR methods over the *LIN* dataset.

side the random node selection method (RAND) for violation resolution. For the HM method a window size of 10 and an approximation order of 1 are employed for the Savitzky-Golay filter.



(a) Communication cost of methods GM and HM for the *LIN* dataset.

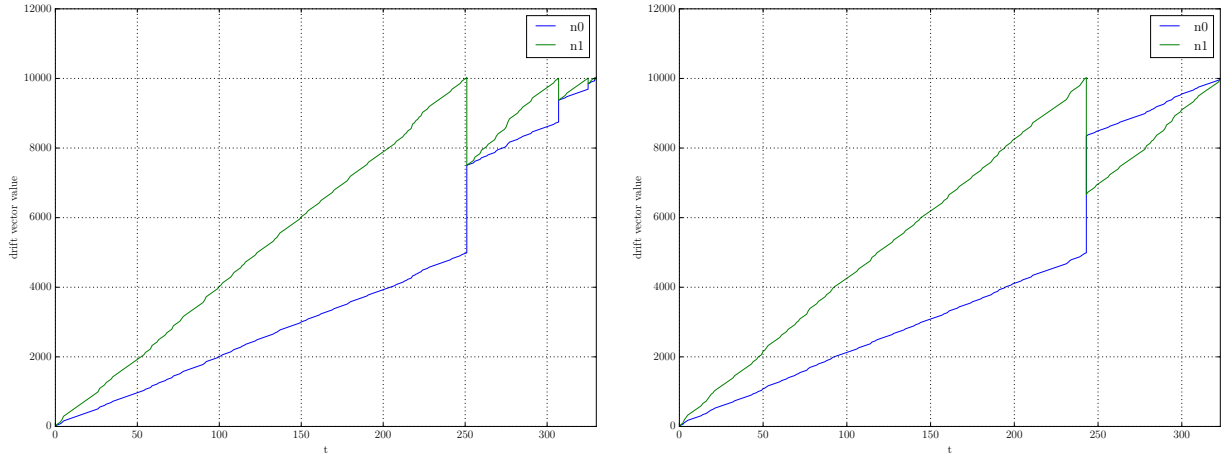
(b) Communication cost of methods GM and HM for the *INT* dataset.

(c) Communication cost of methods GM and HM for the *NOISE* dataset.

Figure 6.9: Comparison of the GM and HM methods in terms of communication cost over a range of 20 nodes. For the Savitzky-Golay filter of HM method a window size of 10 and an order of 1 are employed.

When applied to the *LIN* dataset the HM method consistently surpasses the original GM balancing method in terms of communication reduction by optimally positioning the drift vectors following a Local Violation resolution, a property illustrated in Figure 6.10 for 2 monitoring nodes. When data streams become more complex and unpredictable, i.e., the *INT* and *NOISE* datasets, the HM method provides a similar to slightly worse performance than that of GM, due to variable velocities and accelerations affecting the performance of the optimization function. Additionally,

the selection of random nodes at each resolution attempt hampers the performance of HM due to the fact that optimal positioning computation is relative to the nodes taking part in a specific balancing process. Thus, when an optimized drift vector participates in a subsequent balancing process with a different node set the existing optimization and the "correlation" between optimized sets is discarded.



(a) Drift vectors of 2 nodes, as formulated by the GM algorithm.

(b) Drift vectors of 2 nodes, as formulated by the HM algorithm.

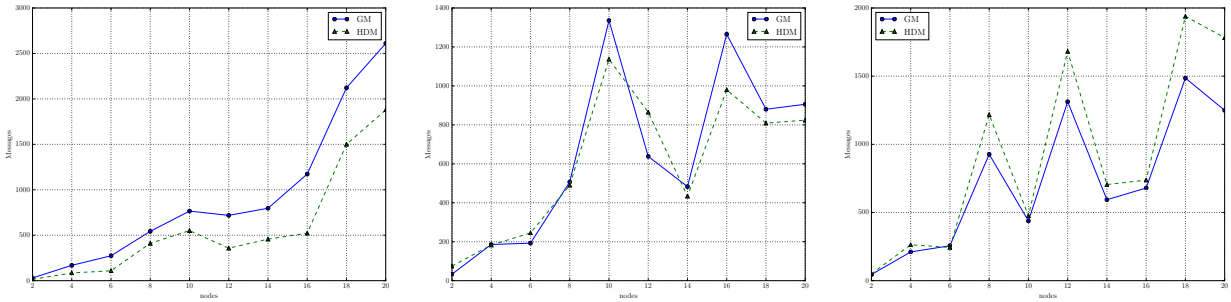
Figure 6.10: The drift vectors of 2 nodes with streams originating from the *LIN* dataset, when the GM and HM balancing methods are applied.

6.2.3 Monitoring Synthetic Data

In order to eliminate the unpredictability the RAND method incurs to HM, we compare the GM method with a combination of our proposed methods (DIST, HM), resulting to the HDM method. Figure 6.11 depicts the performance of the two methods over a range of 2 to 20 nodes, in terms of communication cost. Following that, Figures 6.13 and 6.12 examine the way the Savitzky-Golay filter parameters affect the performance of HDM. Finally, Figure 6.14 illustrates the scalability of the HDM method compared to the GM method, in terms of communication cost and stream dimensionality.

Once again, the HDM method outperforms the original GM method when applied onto the *LIN* dataset for any parameter setting of the Savitzky-Golay filter by achieving a communication

reduction of up to 60 percent. When datasets in the likes of *INT* and *NOISE* are employed, where smooth and drastic velocity changes can be observed, respectively, fine-tuning the filter's parameters can reduce the communication burden of HDM significantly, surpassing the GM algorithm in terms of communication reduction, as shown in Figures 6.13 and 6.12. Finally, when multiple stream dimensions are monitored the HDM method illustrates similar performance as with its one-dimensional stream counterparts.

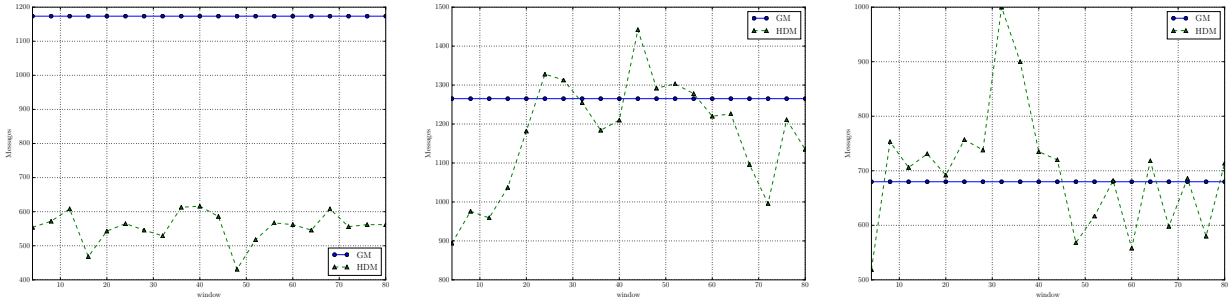


(a) Communication cost of methods GM and HDM for the *LIN* dataset. The Savitzky-Golay window size is set to 10 and the approximation order is set to 1.

(b) Communication cost of methods GM and HDM for the *INT* dataset. The Savitzky-Golay window size is set to 10 and the approximation order is set to 1.

(c) Communication cost of methods GM and HDM for the *NOISE* dataset. The Savitzky-Golay window size is set to 24 and the approximation order is set to 1.

Figure 6.11: Comparison of the GM and HDM methods in terms of communication cost over a range of 20 nodes.

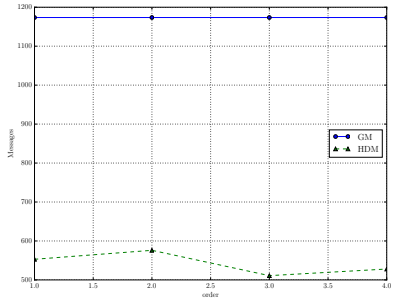


(a) Communication cost of methods GM and HDM for the *LIN* dataset.

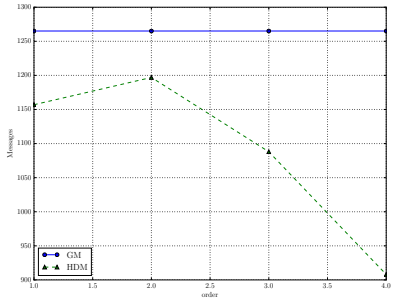
(b) Communication cost of methods GM and HDM for the *INT* dataset.

(c) Communication cost of methods GM and HDM for the *NOISE* dataset.

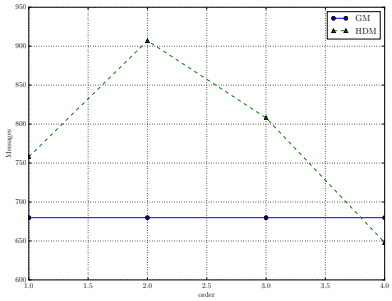
Figure 6.12: Comparison of the GM and HDM methods in terms of communication cost over a range of window sizes, for 16 nodes. The approximation order is set to 1 for all experiments.



(a) Communication cost of methods GM and HDM for the *LIN* dataset. The Savitzky-Golay window size is set to 10.

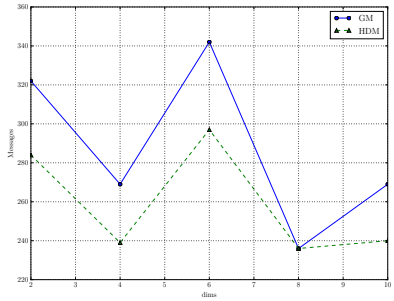


(b) Communication cost of methods GM and HDM for the *INT* dataset. The Savitzky-Golay window size is set to 40.

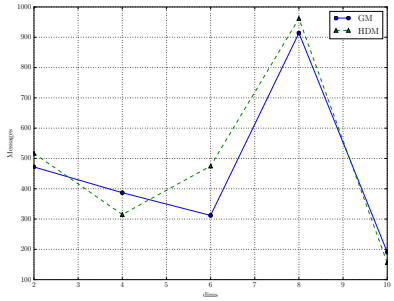


(c) Communication cost of methods GM and HDM for the *NOISE* dataset. The Savitzky-Golay window size is set to 24.

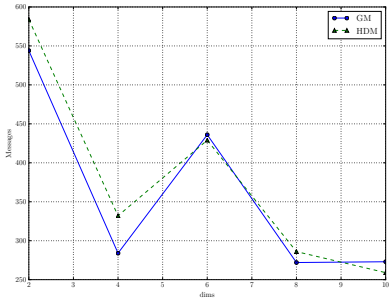
Figure 6.13: Comparison of the GM and HDM methods in terms of communication cost over a range of Savitzky-Golay appriximation orders, for 16 nodes.



(a) Communication cost of methods GM and HDM for the *LIN* dataset.



(b) Communication cost of methods GM and HDM for the *INT* dataset.



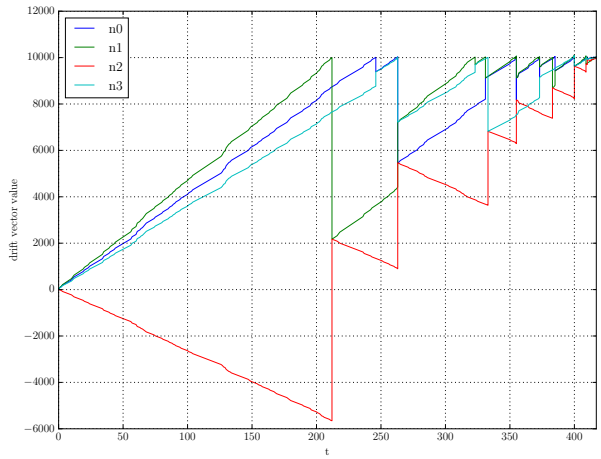
(c) Communication cost of methods GM and HDM for the *NOISE* dataset.

Figure 6.14: Comparison of the GM and HDM methods in terms of communication cost over a range of stream dimensions, for 8 nodes. Savitzky-Golay's window size is set to 10 and the approximation order is set to 1 for all experiments. The monitoring function is a multi-variable quadratic function dependent on the stream dimensionality.

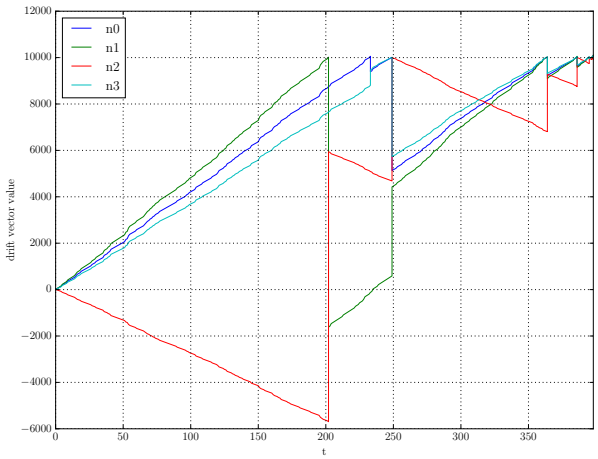
6.2.4 Monitoring Air Quality Data

In this final experiment a real-world dataset is employed, originating from air quality measurements. Specifically, the variance of NO_2 and the ratio of NO to NO_2 are monitored in order to evaluate the HDM and GM methods over real-world applications, once again in terms of communication costs over a range of 4 to 16 nodes, as shown in Figure 6.16.

The HDM method exhibits a decent performance when the appropriate Savitzky-Golay parameters are carefully selected, even though the streams are characterized by great variance and irregularities in time. Compared to the GM method, the HDM method succeeds at reducing the



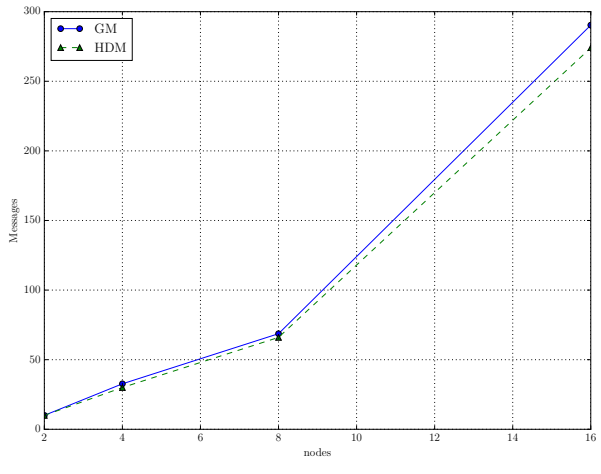
(a) Drift vectors of 2 nodes, as formulated by the GM algorithm.



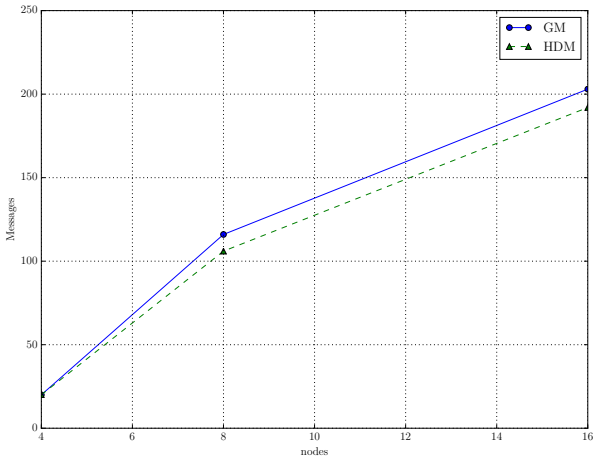
(b) Drift vectors of 2 nodes, as formulated by the HDM algorithm.

Figure 6.15: The drift vectors of 2 nodes with streams originating from the *LIN* dataset, when the GM and HDM methods are applied.

communication overhead by approximately 10 percent.



(a) Communication costs of methods GM and HDM over a range of 4 to 16 nodes, when variance monitoring of NO_2 is taking place. The Savitzky-Golay window size is set to 6 and the approximation order is set to 2.



(b) Communication costs of methods GM and HDM over a range of 4 to 16 nodes, when monitoring the ratio NO to NO_2. The Savitzky-Golay window size is set to 10 and the approximation order is set to 1.

Figure 6.16: Comparison of HDM and GM methods in terms of communication cost over a range of 4 to 16 nodes originating from the air pollution dataset.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

problem statement in short

what has been done in short

our contributions

mop sg distance based node matching directly applicable with the rest of the work done on GM synthetic and actual data explored node scalability oprimal selection of sg parameters good performance on linear and smooth steams short explanation of contributions

7.2 Future Work

parameter estimation algorithm for sg sophisticated prediction models gaussian processes different solvers apply to the rest of the work to explore interraction

References

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ser. PODS '02. New York, NY, USA: ACM, 2002, pp. 1–16. [Online]. Available: http://doi.acm.org/10.1145/543613.543615

[2] I. Sharfman, A. Schuster, and D. Keren, "A geometric approach to monitoring threshold functions over distributed data streams," in *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '06. New York, NY, USA: ACM, 2006, pp. 301–312. [Online]. Available: http://doi.acm.org/10.1145/1142473.1142508

[3] M. N. Garofalakis, D. Keren, and V. Samoladas, "Sketch-based geometric monitoring of distributed stream queries." *PVLDB*, vol. 6, no. 10, pp. 937–948, 2013. [Online]. Available: http://dblp.uni-trier.de/db/journals/pvldb/pvldb6.html#GarofalakisKS13

[4] A. Savitzky and M. J. E. Golay, "Smoothing and differentiation of data by simplified least squares procedures." *Analytical Chemistry*, vol. 36, no. 8, pp. 1627–1639, 1964. [Online]. Available: http://dx.doi.org/10.1021/ac60214a047

[5] J. Edmonds, "Paths, trees and flowers," *Canadian Journal of Mathematics*, pp. 449–467, 1965.

[6] M. Dilman and D. Raz, "Efficient reactive monitoring," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 4, pp. 668–676, May 2002.

[7] R. Keralapura, G. Cormode, and J. Ramamirtham, "Communication-efficient distributed monitoring of thresholded counts," in *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '06. New York, NY, USA: ACM, 2006, pp. 289–300. [Online]. Available: http://doi.acm.org/10.1145/1142473.1142507

[8] C. Olston, J. Jiang, and J. Widom, "Adaptive filters for continuous queries over distributed data streams," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '03. New York, NY, USA: ACM, 2003, pp. 563–574. [Online]. Available: http://doi.acm.org/10.1145/872757.872825

[9] P. B. Gibbons and S. Tirthapura, "Estimating simple functions on the union of data streams," in *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, ser. SPAA '01. New York, NY, USA: ACM, 2001, pp. 281–291. [Online]. Available: http://doi.acm.org/10.1145/378580.378687

[10] B. P. Gibbons and S. Tirthapura, "Distributed streams algorithms for sliding windows," *Theory of Computing Systems*, vol. 37, no. 3, pp. 457–478, 2004. [Online]. Available: http://dx.doi.org/10.1007/s00224-004-1156-4

[11] B. Babcock and C. Olston, "Distributed top-k monitoring," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '03. New York, NY, USA: ACM, 2003, pp. 28–39. [Online]. Available: http://doi.acm.org/10.1145/872757.872764

[12] G. Cormode and M. Garofalakis, "Approximate continuous querying over distributed streams," *ACM Trans. Database Syst.*, vol. 33, no. 2, pp. 9:1–9:39, Jun. 2008. [Online]. Available: http://doi.acm.org/10.1145/1366102.1366106

[13] D. Keren, G. Sagy, A. Abboud, D. Ben-David, I. Sharfman, and A. Schuster, "Safe-zones for monitoring distributed streams," in *Proceedings of the First International Workshop on Big Dynamic Distributed Data, Riva del Garda, Italy, August 30, 2013*, 2013, pp. 7–12. [Online]. Available: http://ceur-ws.org/Vol-1018/paper11.pdf

[14] V. Samoladas, "Unification fo safe zones and the geometric method with application to generalized median monitoring," October 2013.

[15] D. Keren, I. Sharfman, A. Schuster, and A. Livne, "Shape sensitive geometric monitoring," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 8, pp. 1520–1535, Aug 2012.

[16] D. Keren, G. Sagy, A. Abboud, D. Ben-David, A. Schuster, I. Sharfman, and A. Deligiannakis, "Geometric monitoring of heterogeneous streams." *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 8, pp. 1890–1903, 2014. [Online]. Available: http://dblp.uni-trier.de/db/journals/tkde/tkde26.html#KerenSABSSD14

[17] N. Giatrakos, A. Deligiannakis, M. Garofalakis, I. Sharfman, and A. Schuster, "Prediction-based geometric monitoring over distributed data streams," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '12. New York, NY, USA: ACM, 2012, pp. 265–276. [Online]. Available: http://doi.acm.org/10.1145/2213836.2213867

[18] B. Babalis, "A simulator for monitoring data streams," Master's thesis, Technical University of Crete, Chania, Greece, 2013.

[19] D. Ben-David, D. Keren, and A. Schuster, "Violation resolution in distributed stream networks," Master's thesis, Technion University, 2012.

[20] networkx-maximum weight matching. https://networkx.github.io/documentation/networkx-1.9/reference/generated/networkx.algorithms.matching.max_weight_matching.html. Accessed: 2016-02-25.

[21] pyopt - slsqp. http://www.pyopt.org/reference/optimizers.slsqp.html. Accessed: 2016-02-25.

[22] pyopt - conmin. http://www.pyopt.org/reference/optimizers.conmin.html. Accessed: 2016-02-25.

[23] European environmental agency - aq e-reporting. http://www.eea.europa.eu/data-and-maps/data/aqereporting-1.