

**Data Reduction and GPS-Free Node Localization In  
Wireless Sensor Networks**

**D I S S E R T A T I O N**

for the Degree of

Doctor of Philosophy (Computer Science)

**Hüseyin Akcan**

January 2008

**Data Reduction and GPS-Free Node Localization In  
Wireless Sensor Networks**

**D I S S E R T A T I O N**

Submitted in Partial Fulfillment  
of the Requirement for the  
Degree of

**Doctor of Philosophy (Computer Science)**

at the

**POLYTECHNIC UNIVERSITY**

by

**Hüseyin Akcan**

**January 2008**

Approved:



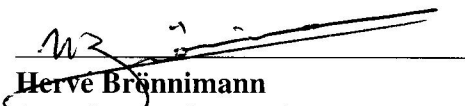
Department Head

12/6/07 2007

Copy No. \_\_\_\_\_


Approved by the Guidance Committee:

Major: Computer Science

  
**Herve Brännimann**  
Associate Professor of  
Computer and Information Science


11/29/07

Date

  
**Alex Delis**  
Associate Professor of  
Computer and Information Science

11/23/07

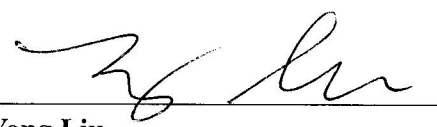
Date

  
**Nasir Memon**  
Professor of  
Computer and Information Science

12/03/07

Date

Minor: Electrical Engineering

  
**Yong Liu**  
Assistant Professor of  
Electrical and Computer Engineering

12/05/2007

Date

Microfilm or other copies of this dissertation are obtainable from

UMI Dissertation Publishing  
Bell & Howell Information and Learning  
300 North Zeeb Road  
P. O. Box 1346  
Ann Arbor, Michigan 48106-1346

## VITA

**Hüseyin Akcan** was born in Mersin, Turkey on November 2, 1977. He received his B.S. degree in Computer and Control Engineering from Istanbul Technical University, Istanbul, Turkey, in 1999. He received his M.S. degree in Computer and Information Science from Polytechnic University, Brooklyn, NY, in 2005. He's been a Ph.D. student in Computer and Information Science Department at Polytechnic University since 2002, and he's been working under the supervision of Prof. Hervé Brönnimann since 2003. He received full fellowship from Polytechnic University during his Ph.D. studies. His research interests include deterministic sampling of count data, data aggregation and node localization in wireless sensor networks.

*To my family  
for their love and support*

## ACKNOWLEDGEMENT

I would like to thank my advisor Prof. Hervé Brönnimann for his invaluable support, guidance, and friendship throughout my studies. He has been an excellent supervisor with his in-depth knowledge, intelligence, and constant encouragement during the preparation of this thesis. I am very grateful for his patience and all his time spent on enlightening discussions.

I would also like to thank Prof. Alex Delis for encouraging me to be more creative and productive in research, as well as for his support and friendship throughout the years.

I am very grateful to Prof. Nasir Memon and Prof. Yong Liu for accepting to be a part of the committee, I feel honored by their presence.

As a student in Polytechnic University, I enjoyed the company of many good friends and I would like to thank all of them, especially Utku Irmak, Levent Şendur, Sertaç Artan, Melda Yüksel, Vassil Kriakov, Yen-Yu Chen, Onur Şahin, Yağız Sütçü, Ayşegül Ergin, Mesut Ali Ergin, Nurcan Tezcan, Hüsrev Taha Sencar, Amitabha Bagchi, Alexander Markowetz, Senem Acet Coşkun, Barış Coşkun, İlker Bayram, Sevinç Bayram, Kaan Bakanoğlu, and Demet Mantar. I am very grateful to all of them for all the conversations during long lunch breaks, even longer coffee breaks, picnics, ski trips etc.

Finally, I thank my mother Nermin, my father Mehmet, and my sister Pınar for their patience and support for all these years. I also thank all my mentors and teachers, especially my elementary school teacher Şengül Külcü for giving me the joy of learning from the very beginning.

This work is supported by National Science Foundation CAREER GRANT CCR-0133599.

# Contents

<b>List of Figures</b>	<b>iv</b>
<b>List of Tables</b>	<b>vii</b>
<b>1 Introduction</b>	<b>3</b>
<b>2 Sampling Algorithms For Count Data</b>	<b>6</b>
2.1 Related work . . . . .	8
2.2 Deterministic sampling algorithms . . . . .	10
2.2.1 Notation . . . . .	10
2.2.2 Deterministic Reservoir Sampling (DRS) . . . . .	11
2.3 Experimental results . . . . .	15
2.3.1 Datasets used . . . . .	15
2.3.2 Sampling count data . . . . .	16
2.3.3 Extensions to DRS algorithm . . . . .	22
2.4 Concluding remarks . . . . .	27
<b>3 Distributed Sampling Algorithms For Sensor Networks</b>	<b>28</b>
3.1 Related work . . . . .	31
3.2 Distributed deterministic sampling . . . . .	34
3.2.1 Notation . . . . .	35
3.2.2 Aggregation data structure . . . . .	35
3.2.3 Motivation for weighted sampling . . . . .	36
3.2.4 Deterministic Weighted Sampling (DWS) . . . . .	37
3.3 Experiments . . . . .	41
3.3.1 Climate dataset . . . . .	41
3.3.2 Distributed data reduction . . . . .	42
3.3.3 Example SQL queries . . . . .	48
3.4 Concluding remarks . . . . .	53
<b>4 GPS &amp; Compass Free Node Localization On Wireless Sensor Networks</b>	<b>54</b>
4.1 Related work . . . . .	56
4.2 Localization algorithms . . . . .	58
4.2.1 GPS-Free Directed Localization (GDL) . . . . .	59



4.2.2	GPS and Compass-Free Directed Localization (GCDL) . . . . .	64
4.3	An example sensor network mobility algorithm . . . . .	70
4.4	Experiments . . . . .	72
4.4.1	Evaluation of the GDL algorithm . . . . .	72
4.4.2	Evaluation of the GCDL algorithm . . . . .	76
4.4.3	Comparison with an absolute positioning algorithm . . . . .	79
4.5	Concluding remarks . . . . .	83
<b>5</b>	<b>Conclusion</b>	<b>88</b>
	<b>Bibliography</b>	<b>90</b>

## List of Figures

2.1	The DRS algorithm . . . . .	13
2.2	Dataset parameters . . . . .	15
2.3	RMS error of SRS, EASE, Biased-L2, and DRS for four datasets (BMS1, T5, T10, T50). . . . .	17
2.4	Accuracies of SRS, EASE, Biased-L2, and DRS for four datasets (BMS1, T5, T10, T50). . . . .	18
2.5	Ratio of the RMS error of SRS over EASE, Biased-L2, and DRS for four datasets (BMS1, T5, T10, T50). Higher $y$ -coordinate values correspond to better quality samples. . . . .	19
2.6	Ratio of the accuracy of EASE, Biased-L2, and DRS over SRS for four datasets (BMS1, T5, T10, T50). Higher $y$ -coordinate values correspond to more accurate samples . . . . .	20
2.7	Time spent per transactions (in milliseconds) for each algorithm, with various sampling rates. . . . .	21
2.8	Effect of $k$ on sample quality, $ratio =  S_{DRS} /k$ . . . . .	22
2.9	Accuracy and CPU time vs. sample rate for Biased-L2 and DRS with different $k$ values (left, right). . . . .	23
2.10	RMS error vs. number of elements processed, while (left) increasing and (right) decreasing the sample size suddenly after 30 000 transactions. . . . .	24
2.11	Plot of RMS error vs. number of transactions examined while converting random to deterministic samples for different sample sizes. . . . .	25
2.12	Effect of changing the $k$ parameter on the speed of the DRS algorithm. Synthetic dataset (T10I6D100K) (left), and real-world dataset (BMS1) (right). For both datasets selecting $k \approx 25$ seems quite reasonable. The time spend per transaction on each dataset varies with the average number of items per transaction. The synthetic dataset we test has more items per transaction on average than the real-world (BMS1) dataset, which increases the time per transaction on the synthetic dataset. . . . .	26
3.1	An example aggregation tree showing the nodes, samples and the weights for each sample. Each node gathers samples from its own data and children's samples and creates a sample of size $s$ . The sampling process is well distributed on each node, since each node maintains a sample of size $s$ . . . . .	37

3.2	The Deterministic Weighted Sampling algorithm . . . . .	39
3.3	Average values of precipitation (top left), snow fall (top right), snow amount (center left), maximum temperature (center right), and minimum temperature (bottom). . . . .	43
3.4	Time period for weather data used in the simulations. . . . .	44
3.5	Simulated U.S. area (60x25), and 227 sensor locations. . . . .	44
3.6	Results for real world climate dataset. (a) RMS results vs. sample size, (b) energy usage vs. sample size, and (c) energy usage vs. sample quality. . . .	46
3.7	Results for synthetic dataset. (a) RMS results vs. sample size, (b) energy usage vs. sample size, and (c) energy usage vs. sample quality. . . . .	47
3.8	SQL query errors for deterministic sample and DIR samples. Results for Query-1 (top left), Query-2 (top right), Query-3 (bottom left) and Query-4 (bottom right). . . . .	52
4.1	(a) Typical movement of two nodes, with angles and distances. (b) An example non-rigid geometry, where nodes move an equal distance in parallel. In the <i>equal parallel movement</i> exceptional configuration, localization is not possible because, geometrically, nodes can have infinite positions around each other. . . . .	60
4.2	Core localization algorithm for $n_1$ : calculates two possible positions for $n_2$ . Verification algorithm evaluates the position estimations of neighbor nodes such that only 1 out of 4 position pairs validates the distance. . . . .	62
4.3	2-step motion for localizing the <i>blue</i> node. . . . .	65
4.4	Geometry of the GCDL localization showing the <i>blue</i> (dark) node stationary at position $x_3, y_3$ , and <i>red</i> (light) node performing 2-step motion from $x_1, y_1$ to positions $x_0, y_0$ and $x_2, y_2$ in order to localize the <i>blue</i> node. . . . .	65
4.5	Zig-zag motion of nodes towards movement direction. . . . .	68
4.6	Detecting and correcting the skew between local coordinate systems of neighbors. When the relative positions of the nodes to each other are $a$ and $b$ , the skew in their local coordinate systems is $\alpha$ . . . . .	68
4.7	The mobility algorithm we use for directed motion. $RF$ is the fraction of the wireless range; used by nodes as an ideal distance with their neighbors, and $range(NodeN)$ is the wireless range of the given node. . . . .	71
4.8	Percent of non-localized nodes for different node densities. . . . .	73
4.9	Effects of angle and distance measurement noise on position error (a), and percent of non-localized nodes (b), for GDL . . . . .	74
4.10	Mean and standard deviation of position error of GDL vs. speed of nodes with a wireless range of 10 units and speed measured in units per epoch. . .	76
4.11	Effect of number of epochs on selected common north angle, (a) random motion, and (b) directed motion, for GCDL . . . . .	77

4.12	Effects of angle and distance measurement noise on selected common north error in degrees, (a) random motion, and (b) directed motion, for GCDL. The distance and angle noise axes show the percent error over the actual measurement. . . . .	78
4.13	Mean and standard deviation of position error vs. number of epochs for our algorithms and the absolute positioning algorithm performing random movement, using different levels of noise. The error of the absolute positioning algorithm increases with the number of epochs while the error of our algorithms are almost constant, which is attributed to the memoryless property of our algorithms. . . . .	80
4.14	Mean and standard deviation of position error vs. number of epochs for our algorithms and the absolute positioning algorithm performing directed movement, using different levels of noise. The error of the absolute positioning algorithm increases with the number of epochs while the error of our algorithms is almost constant, which is attributed to the memoryless property of our algorithms. . . . .	81
4.15	Average time in milliseconds (ms) spent for localization calculations in core GDL and GCDL algorithms. . . . .	82
4.16	Directed trajectory of nodes performing zig-zag movement. . . . .	83
4.17	Snapshots of absolute positioning algorithm performing directed motion in Figure 4.16. . . . .	84
4.18	Snapshots of GDL algorithm performing directed motion in Figure 4.16. . .	85
4.19	Snapshots of GCDL algorithm performing directed motion in Figure 4.16. .	86

## List of Tables

3.1	Ratio of the mean, min and max error values of the DIR samples to the error values of DWS sample for each query. . . . .	51
-----	--	----

## **AN ABSTRACT**

### **Data Reduction and GPS-Free Node Localization In Wireless Sensor Networks**

**by**

**Hüseyin Akcan**

**Advisor: Hervé Brönnimann**

Submitted in Partial Fulfillment of the Requirements  
for the Degree of Doctor of Philosophy (Computer Science)

January 2008

This thesis addresses the topics of data reduction via sampling in both central database environments, and wireless sensor networks, and GPS-free node localization in wireless sensor networks. The first contribution of this thesis is a deterministic sampling algorithm for sampling count data, which is common in data mining applications. We show that our algorithm creates more accurate and higher quality samples compared to previous work, and the samples it generates can be used as a surrogate for the original high volume data.

Our second contribution is a deterministic weighted sampling algorithm that can be used as a new data aggregation method for wireless sensor network data. The aggregation algorithm shares similar ideas with our previous sampling algorithm. In order to adapt to the sensor network environment, however, we designed our algorithm to perform weighted sampling in a distributed manner. The weighted sampling design allows the algorithm to work with any arbitrary network topology, while the distributed design divides

the sampling work equally on all the sensor nodes in the network and prevents any node from being a bottleneck (both with regards to CPU consumption, and communication). We show that our aggregation algorithm generates samples of better quality than previous algorithms, using far less energy.

Our last contribution is two GPS-free node localization algorithms, termed GPS-free Directed Localization (GDL), and GPS & Compass-free Directed Localization (GCDDL). The importance of localization is apparent in mobile wireless sensor networks, where the neighborhood changes frequently and knowledge about the neighbor positions is essential for performing additional tasks such as aggregation or coherent movement. Our algorithms perform localization without the need of Global Positioning System (GPS) or any other infrastructure (e.g., anchor points). These algorithms work with only local knowledge without using historical data, and exploit mobility to perform localization. The memoryless aspect of our algorithms avoids the accumulation error over time, which is essential in mobility scenarios where coherent movement of a swarm of nodes is required. We show that our algorithms do work even at high environmental noise levels, and keep a nice semi-rigid network formation in mobility scenarios.

# **Chapter 1**

## **Introduction**

In this thesis we investigate three different topics in computer science, and propose novel methods and algorithms. Each of these algorithms addresses a specific problem in its domain that we present in detail in the following chapters. The presentation order of these algorithms follow the chronological order of the work, as well as a logical chain of ideas. Here we describe each of these algorithms briefly, while trying to highlight the connections between each one of them.

Processing and extracting meaningful knowledge from count data is an important problem in data mining. The volume of data is increasing dramatically as the data is generated by day-to-day activities such as market basket data, web clickstream data or network data. Most mining and analysis algorithms require multiple passes over the data, which requires extreme amounts of time. One solution to save time would be to use samples, since sampling is a good surrogate for the data and the same sample can be used to answer many kinds of queries. To address this problem, we developed a deterministic sampling algorithm, DRS, that produces samples vastly superior to the previous deterministic and random algorithms, both in sample quality and accuracy. In Chapter 2 we discuss the problem, related work and our algorithm in detail, also present experimental results comparing our algorithm to previous work. The main product of this research is presented in [4, 3].

Having described the benefits of data reduction in central databases, we can extend this approach to wireless sensor networks, as a way of doing in network data aggre-



gation. The processing capabilities of wireless sensor nodes enable to aggregate redundant data to limit total data flow over the network. The main property of a good aggregation algorithm is to extract the most representative data by using minimum resources. From this point of view, sampling is a promising aggregation method as it represents the whole data, and once extracted can be used to answer multiple kinds of queries (such as AVG, MEDIAN, SUM, COUNT, etc.), at no extra cost to the sensor network. Additionally, sampling also preserves correlations between attributes of multi-dimensional data, which is quite valuable for further data mining. In Chapter 3, we describe a distributed weighted sampling algorithm to sample sensor network data and compare to an existing random sampling algorithm, which is the only algorithm to work in this kind of setting. We perform popular queries to evaluate our algorithm on a real world data set, which covers climate data in the U.S. for the past 100 years. During testing, we focus on issues such as sample quality, network longevity, energy and communication costs. The main product of this research is presented in [5, 6].

In order to extend in-network data aggregation to mobile nodes, we need dynamic aggregation structures to adapt to continuously changing topologies. One way to efficiently perform this task is to have an easy way to access location information of mobile nodes, therefore in Chapter 4 we describe two localization algorithms that suit this requirement. An important problem in mobile ad-hoc wireless sensor networks is the localization of individual nodes, i.e., each node's awareness of its position relative to the network. We introduce a variant of this problem *directional* localization where each node must be aware of both its position *and* orientation relative to its neighbors. This variant is especially relevant for the applications in which mobile nodes in a sensor network are required to move in a collaborative manner. Using global positioning systems for localization in large scale sensor networks is not cost effective and may be impractical in enclosed spaces. On the other hand, a set of pre-existing anchors with globally known positions may not always be available. To address these issues, in Chapter 4 we propose two algorithms for directional node localization based on relative motion of neighboring nodes in an ad-

hoc sensor network without an infrastructure of global positioning systems (GPS), anchor points, or even mobile seeds with known locations. Our first algorithm, GPS-free Directed Localization (GDL) assumes the availability of a digital compass on each sensor node. We relax this requirement in our second algorithm termed GPS and Compass free Directed Localization (GCDL). Through simulation studies, we demonstrate that our algorithms scale well for large numbers of nodes and provide convergent localization over time, even with errors introduced by motion actuators and distance measurements. Furthermore, based on our localization algorithms, we introduce mechanisms to preserve network formation during directed mobility in sensor networks, which is important for applications that require good area coverage and coherent movement. Our simulations confirm that, in a number of realistic scenarios, our algorithms provide for a mobile sensor network that preserves its formation over time, irrespective of speed. Also the on demand behavior of our algorithms avoids storing any long term historical movement information, which keeps our localization algorithms free from cumulative errors. The main product of this research is presented in [8, 9].

## Chapter 2

### Sampling Algorithms For Count Data

Count data serve as the input for an important class of online analytical processing (OLAP) tasks, including association rule mining[2] and data cube online exploration[35]. These data are often stored in databases for further processing. However, the volume of data has become so huge that mining and analysis algorithms that require several passes over the data are becoming prohibitively expensive. Sometimes, it is not even feasible (or desirable) to store it in its entirety, e.g., with network traffic data. In that case, the data must be processed as a stream. For most OLAP tasks, exact counts are not required and an approximate representation is appropriate, motivating an approach called *data reduction*[12]. A similar trend was observed in traditional database management systems (DBMS) where exact results taking too long led to approximate query answering as an alternative[40, 33].

A general data reduction approach that scales well with the data is sampling. The data stream community also uses sampling as a representative for streaming data [10, 46]. Even though sampling is widely used for analyzing data, the use of random samples can lead to unsatisfactory results. For instance, samples may not accurately represent the entire data due to fluctuations in the random process. This difficulty is particularly apparent for small sample sizes and bypassing it requires further engineering.

The main product of this research consists of a deterministic sampling algorithm, named below DRS, to find a sample  $S$  from a dataset  $D$  which optimizes the root mean square (RMS) error of the frequency vector of items over the sample (when compared to

the original frequency vector of items in  $D$ ). DRS is a clear improvement over SRS (simple random sample) and other more specialized deterministic sampling algorithms such as FAST[22] and EASE[16]. The samples our algorithm produces can be used as surrogate for the original data, for various purposes such as query optimization, approximate query answering[33, 40], or further data mining (e.g., building decision trees or iceberg cubes). In this latter context, the items represent all the values of all the attributes in the DBMS and one wants to maintain, for each table, a sample which is representative for every attribute simultaneously. We assume here categorical attributes—numerical attributes can be discretized, e.g., by using histograms and creating a category for each bucket.

In Section 2.1 we talk about the previous work. Later, in Section 2.2 we present our sampling algorithm, Deterministic Reservoir Sampling (DRS), for deterministically sampling count data. In Section 2.3 we evaluate DRS on several real-world and synthetic datasets, with various criteria and settings. Finally, in Section 2.4 we finish with the concluding remarks.

## Our Contributions

- We present a deterministic sampling algorithm: DRS, to sample count data (tabular or streaming).
- Our algorithm generates samples with better accuracy and quality compared to the previous algorithms (EASE and SRS).
- DRS improves on previous algorithms both in run-time and memory footprint.
- We perform extensive simulations with synthetic and real-world datasets under various settings, and demonstrate the superiority of our algorithm.

## 2.1 Related work

The survey by Olken and Rotem[60] gives an overview of random sampling algorithms in databases. Sampling is discussed and compared against other data reduction methods in the NJ Data Reduction Report[12]. In addition to sampling, a huge literature is available on histograms[37] and wavelet decompositions as data reduction methods, and we do not attempt to survey it here. We note however that sampling provides a general-purpose reduction method which simultaneously applies to a wide range of applications. Moreover, the benefits of sampling vs. other data reduction methods are increased with multi-dimensional data: the larger the dimension, the more compact sampling becomes vs., e.g., multi-dimensional histograms or wavelet decompositions[12]. Also, sampling retains the relations and correlations between the dimensions, which may be lost by histograms or other reduction techniques. This latter point is important for data mining and analysis.

Zaki *et al.*[70] state that simple random sampling can reduce the I/O cost and computation time for association rule mining. Toivonen[66] propose a sampling algorithm that generates candidate itemsets using a large enough random sample, and verifies these itemsets with another full database scan. Instead of a static sample, John and Langley[45] use a dynamic sample, where the size is selected by how much the sample represents the data, based on the application. The FAST algorithm introduced by Chen *et al.*[22] creates a deterministic sample from a relatively large initial random sample by trimming or growing a sample according to a local optimization criterion. The EASE algorithm by Brönnimann *et al.*[16] again uses a relatively large sample and creates a deterministic sample by performing consecutive halving rounds on the sample. EASE algorithm keeps penalty functions for each item per each separate halving round. Each transaction has to pass the test at each level in order to be added to the sample. The penalties change based on the accept or reject decision of the transaction, and the goal is to generate a sample having item supports as close as possible to those in the dataset. Multiple halving rounds per transaction and the penalty functions used for each round introduces additional complexity to EASE compared

to Biased-L2. The Biased-L2 algorithm uses ideas similar to EASE based on discrepancy theory[21], but samples the dataset without introducing halving rounds and improves on the run-time and memory requirements, as well as the sample quality and accuracy. Biased-L2 algorithm is generic for any discretized data, and in [7] it is applied to sampling geometric point data for range counting applications.

The main difference between DRS and FAST is that DRS keeps a smaller sample in memory, examines each transaction only once, and it is suitable to handle streaming data. DRS algorithm uses a cost function based on RMS distance which is incrementally updated by changes to the sample. In this work we only give the incremental formulas specific to our case, where the sample size does not change by updates. Additional incremental formulas for various distance functions are presented in [15]. As the sample size can be preset exactly, DRS does not have accuracy problems caused by the halving rounds of EASE. Johnson *et al.*[46] suggest that, for cases when the stream size is unknown, it is useful to keep a fixed-sized sample. Since in practice most stream sizes are unknown, this can be best done by allowing the algorithms to dynamically remove transactions from the sample, as in reservoir sampling[67] and DRS.

Gibbons *et al.*[31] propose concise sampling and introduce algorithms to incrementally update a sample for any sequence of deletions and insertions. While concise sample dramatically reduces memory footprint, it works for single attribute sampling, lacking the ability to give any correlation between attributes, which is desirable for multi-dimensional data.

Vitter[67] introduces reservoir sampling, which allows random sampling of streaming data. Reservoir sampling produces a sample of quality identical to SRS, but does not examine all the data. Whenever a new record is selected, it evicts a random record. In contrast, DRS adapts to changes in distribution by deterministically selecting the worst record to evict. Gibbons *et al.*[32] use reservoir sampling as a backing sample to keep the histograms up to date under insertions and future deletions.

In [46] different approximation algorithms are discussed including Reservoir

Sampling[67], Heavy Hitters algorithm[58], Min-Hash Computation[27] and Subset-Sum sampling[30]. Among these we only compare our algorithms with Reservoir Sampling (random sampling in general), since the rest of the algorithms are tailored for specific applications.

## 2.2 Deterministic sampling algorithms

In this section we first describe the notation used, and then present our deterministic sampling algorithm: Deterministic Reservoir Sampling, in Section 2.2.2.

### 2.2.1 Notation

Let  $D$  denote the database of interest,  $d = |D|$  the number of transactions,  $S$  a deterministic sample drawn from  $D$ , and  $s = |S|$  its number of transactions. We denote by  $I$  the set of all items that appear in  $D$ , by  $m$  the total number of such items, and by  $\text{size}(j)$  the number of items appearing in a single transaction  $j \in D$ . We let  $T_{\text{avg}}$  denote the average number of items in a transaction, so that  $dT_{\text{avg}}$  denotes the total size of  $D$  (as counted by a complete item per transaction enumeration).

In the context of association rule mining, an *itemset* is a subset of  $I$ , and we denote by  $\mathcal{I}(D)$  the set of all itemsets that appear in  $D$ ; a set of items  $A$  is an element of  $\mathcal{I}(D)$  if and only if the items in  $A$  appear jointly in at least one transaction  $j \in D$ . A  $k$ -itemset is an itemset with  $k$  items, and their collection is denoted by  $\mathcal{I}_{\parallel}(D)$ ; in particular the 0-itemset is the empty set (contained in all the transactions) and the 1-itemsets are simply the original items. Thus  $\mathcal{I}(D) = \cup_{\parallel \geq 0} \mathcal{I}_{\parallel}(D)$ . The itemsets over a sample  $S \subseteq D$  are  $\mathcal{I}(S) \subseteq \mathcal{I}(D)$ , and  $\mathcal{I}_{\parallel}(S)$  is defined similarly.

For a set  $T$  of transactions and an itemset  $A \subseteq I$ , we let  $n(A; T)$  be the number of transactions in  $T$  that contain  $A$  and  $|T|$  the total number of transactions in  $T$ . Then the support of  $A$  in  $T$  is given by  $f(A; T) = n(A; T)/|T|$ . In particular,  $f(A; D) = n(A; D)/|D|$  and  $f(A; S) = n(A; S)/|S|$ . Given a threshold  $t > 0$ , an item is frequent in

$D$  (resp. in  $S$ ) if its support in  $D$  (resp.  $S$ ) is no less than  $t$ .

The distance between two sets  $D$  and  $S$  with respect to the 1-itemset frequencies can be computed via the discrepancy of  $D$  and  $S$ , defined as

$$Dist_{\infty}(D, S) = \max_{A \in I} |f(A, D) - f(A, S)|. \quad (2.1)$$

A sample  $S$  such that  $Dist_{\infty}(D, S) \leq \varepsilon$  is called an  $\varepsilon$ -approximation. Other ways to measure the distance of a sample are via the L1-norm or the L2-norm (also called ‘root-mean-square’ - RMS),

$$Dist_1(D, S) = \sum_{A \in I} |f(A, D) - f(A, S)|, \quad (2.2)$$

$$Dist_2(D, S) = \sqrt{\sum_{A \in I} (f(A, D) - f(A, S))^2}. \quad (2.3)$$

In order to measure the accuracy of the sample  $S$  for evaluating frequent itemset mining, as in [22, 16] the following measure is used:

$$Accuracy(S) = 1 - \frac{|L(D) \setminus L(S)| + |L(S) \setminus L(D)|}{|L(S)| + |L(D)|}, \quad (2.4)$$

where  $L(D)$  and  $L(S)$  represent the number of frequent itemsets in dataset and sample.  $L(D) \setminus L(S)$  represents the number of itemsets exist in dataset but not in sample, and  $L(S) \setminus L(D)$  the other way around.

### 2.2.2 Deterministic Reservoir Sampling (DRS)

In this section, we present Deterministic Reservoir Sampling algorithm. The main idea is to maintain a sample of constant size  $s$  and periodically add a transaction while evicting another. The choice of transactions is computed in order to keep a distance function as small as possible (here, we present the algorithm using  $Dist_2$ ). In particular, it differs from EASE and Biased-L2 by its ability to not only add new transactions to the sample but also remove undesired transactions. As we will show, this ability makes the sample more robust to changes of the distribution in the streaming or tabular data scenarios.



The algorithm maintains the *worst* transaction  $W$  in the sample, i.e., the one whose removal from  $S$  decreases  $Dist_2(D, S)$  the most. An *update* by  $T$  consists of replacing  $W$  by some transaction  $T$ . The parameter  $k$  is used to control the number of updates as follows: The algorithm scans the consecutive transactions in blocks of size  $k$  and for each block, computes the best transaction  $T$  for an update, i.e., such that  $Dist_2(D, (S \setminus \{W\}) \cup \{T\})$  is minimized. One important observation here is that even replacing  $W$  by the best  $T$  may not decrease the cost function in some situations. In this case, the sample is kept unchanged for this block.

The full algorithm is presented in Figure 2.1. This version of the algorithm works in a single pass and updates the supports of items on the fly, both in the dataset and in the sample. The only requirement for single pass is that the size of the dataset must be known in advance, in order to compute the  $Dist_2$  function. For the streaming case, or other cases where we do not know the size of the dataset in advance, a slight modification to the algorithm is possible, that starts with an expected dataset size and zero frequencies, and gradually increases them during run-time.

Since at each update, only one transaction is added and another removed from the sample, limited number of items on average are affected from this change, allowing us to easily update the penalty function incrementally. The difference in penalty function after adding transaction  $T$  is:

$$\Delta_T = \frac{\text{size}(T)}{s^2} + \frac{2}{s} \sum_{i \in T} \left( f(A_i, S_{DRS}) - f(A_i, D) \right), \quad (2.5)$$

Similarly, the difference after removing transaction  $W$  is

$$\Delta_W = \frac{\text{size}(W)}{s^2} - \frac{2}{s} \sum_{j \in W} \left( f(A_j, S_{DRS}) - f(A_j, D) \right), \quad (2.6)$$

By adding these two differences together, we can find the total difference caused by the

DRS ( $D, k$ )

```

1:  $S_{DRS} \leftarrow$  first transactions of  $D$ 
2:  $N \leftarrow 0$ ;  $C \leftarrow Dist_2(D, S_{DRS})$ 
3:  $W \leftarrow \text{FINDWORSE}(D, S_{DRS})$ 
4:  $C_{min} \leftarrow \infty$ ;  $T_{min} \leftarrow \emptyset$ 
5: for each transaction  $j$  in  $D$  do
6:    $N \leftarrow N + 1$ 
7:    $C_{new} = Dist_2(D, S_{DRS} \cup \{j\} \setminus \{W\})$ 
8:   if  $C_{new} < C_{min}$  then
9:      $C_{min} \leftarrow C_{new}$ ;  $T_{min} \leftarrow j$ 
10:  end if
11:  if  $N \equiv 0 \pmod k$  then
12:    {periodical updates, after every  $k$  transactions}
13:    if  $C_{min} < C$  then
14:       $S_{DRS} \leftarrow S_{DRS} \cup \{T_{min}\} \setminus \{W\}$ 
15:       $C \leftarrow C_{min}$ 
16:       $W \leftarrow \text{FINDWORSE}(D, S_{DRS})$ 
17:    end if
18:     $C_{min} \leftarrow \infty$ ;  $T_{min} \leftarrow \emptyset$ 
19:  end if
20: end for
21: return  $S_{DRS}$ 

```

FINDWORSE ( $D, S_{DRS}$ )

```

1:  $C_w \leftarrow \infty$ ;  $W \leftarrow \emptyset$ 
2: for each transaction  $j$  in  $S_{DRS}$  do
3:   if  $Dist_2(D, S_{DRS} \setminus \{j\}) < C_w$  then
4:      $W \leftarrow j$ ;  $C_w \leftarrow Dist_2(D, S_{DRS} \setminus \{j\})$ 
5:   end if
6: end for
7: return  $W$ 

```

Figure 2.1: The DRS algorithm

update since the sample size doesn't change:

$$\begin{aligned}
 Dist_2(D, S_{DRS} \cup \{T\} \setminus \{W\}) &= \frac{\text{size}(T) + \text{size}(W)}{s^2} \\
 &+ \frac{2}{s} \sum_{i \in T \setminus W} \left( f(A_i, S_{DRS}) - f(A_i, D) \right) \\
 &- \frac{2}{s} \sum_{j \in W \setminus T} \left( f(A_j, S_{DRS}) - f(A_j, D) \right). \quad (2.7)
 \end{aligned}$$

Based on Equation (2.7) the computation can be done in time  $O(T_{\text{avg}})$  per transaction. Thus the run-time cost of one iteration of the loop is  $O(T_{\text{avg}})$  except if it triggers an update, in which case it becomes  $O(sT_{\text{avg}})$  since each transaction in the sample needs to be re-examined to find the new worse transaction. In order to describe the overall running time, we consider the choice of  $k$ . A choice of  $k = 1$  means that we update in a totally greedy fashion (steepest descent), which might perform well in terms of error but might be very expensive in terms of run-time. A choice of  $k = d$  means no updates. In between these extremes, selecting a bigger value will decrease the number of updates on the sample and speed up the sampling process, but decrease the quality of the sample. Selecting a smaller value will slow down the process while increasing the quality of the sample. Ultimately, we should pick the smallest value of  $k$  which affords a reasonable running time. Following the analogy with reservoir sampling[67], we could hope that the number of updates is  $O(\log d/s)$ , yet our updates are dictated by a comparatively more complex process, and this hope is not borne out by the experiments. Instead, the actual bounds seems closer to the trivial upper bound of  $d/k$ . A good compromise seems  $k = s/c$  for some constant  $c > 0$ , which implies a total number of updates which is  $O(d/s)$  and thus a total run-time of  $O(dT_{\text{avg}})$ . Empirical results are given in the experiments section, showing the effect of  $k$  on the overall sample quality and accuracy.

As for memory requirements, DRS needs to store the frequency counts of every item separately in  $D$  and  $S_{DRS}$ , as well as the sample, hence has a space complexity of  $O(m + sT_{\text{avg}})$ , which is equivalent to that of Biased-L2 (since finally  $s = \alpha d$ ).

Dataset	T5I3D100K	T10I6D100K	T50I10D100K	BMS1
NoOfTrans	100000	100000	100000	59602
NoOfItems	1000	1000	1000	497
AvgTransSize	5	10	50	2.5
Apr.Supp	0.4	0.4	0.75	0.3
1-itemsets	460	633	832	225
2-itemsets	84	774	45352	169
3-itemsets	39	445	40241	39
4-itemsets	18	378	45140	0

Figure 2.2: Dataset parameters

## 2.3 Experimental results

In this section, we compare DRS with the previous ones on various datasets, and show the superiority of our algorithm in terms of sample quality and accuracy. We also highlight additional features of the DRS algorithm using tailored experiments.

### 2.3.1 Datasets used

The datasets used in our experiments are three synthetic datasets from IBM[65] (T5I3D100K, T10I6D100K, T50I10D100K), and one real-world clickstream dataset (BMS-WebView-1 or BMS1)[48]. Both types of datasets are count datasets, with variable length transaction sizes. The detailed parameter information of each dataset is listed in Figure 2.2. The reason for our choice of datasets is to have different maximum lengths of a transaction and of an itemset, in order to evaluate the dependency on these parameters and make sure the results don't differ too much. BMS1 acts as the real-world/typical control data set. More detailed information about the BMS1 dataset can be found in[48].

### 2.3.2 Sampling count data

In this section, we compare the results of the simple random sample (SRS), EASE, Biased-L2, and DRS algorithms on the association rule datasets. The comparison is based on the quality and accuracy of the sample, given the cost function in Eq. (2.3) and the accuracy function in Eq. (2.4).

Figure 2.3 plots the RMS error, and Figure 2.4 the accuracy results of SRS, EASE, Biased-L2, and DRS on all four datasets. The algorithms are run with sampling rates of 0.003, 0.007, 0.015, 0.03, and 0.062, or the sample size equivalent of them, based on the size of the dataset. For synthetic datasets, the algorithms are run 50 times with a random shuffle of  $D$ , and the average is calculated. For the real-world dataset (BMS1), the original order of transactions is kept, and deterministic algorithms (EASE, Biased-L2, and DRS) are run once, while random sampling algorithm (SRS) is again run 50 times.

Figure 2.5<sup>1</sup> presents the ratio comparison of results in Figure 2.3 relative to SRS for each dataset. From these results, we can say that the sample quality of DRS and Biased-L2 are superior compared to the results of EASE and SRS. In terms of RMS error, on average, DRS is a factor of 14 times and Biased-L2 is a factor of 12 times better than EASE on the real-world dataset, and a factor of 2 times better on synthetic datasets. On average, DRS is also a factor of 6 times better than SRS on all datasets.

In order to compare the accuracy of the samples, we use the Apriori[2] algorithm to generate association rules both for the dataset and the samples. Later, we use Equation (2.4) to calculate accuracies. Figure 2.4 plots the accuracy results of SRS, EASE, Biased-L2 and DRS on all datasets. In addition, Figure 2.6<sup>1</sup> presents the average ratio comparison of the accuracy results for each dataset, based on the SRS accuracy for each sampling rate. From the figures we can say that on average, DRS is a factor of 12, and Biased-L2 is a factor of 8 times better than EASE on real-world dataset. Also on this real-world dataset, DRS is a factor of 5 times, and Biased-L2 is a factor of 4 times better than SRS algorithm.

---

<sup>1</sup>EASE algorithm was unable to generate the expected sample sizes on BMS1 dataset, the accuracy and quality ratio values of EASE algorithm on this dataset are linearly estimated based on existing data.

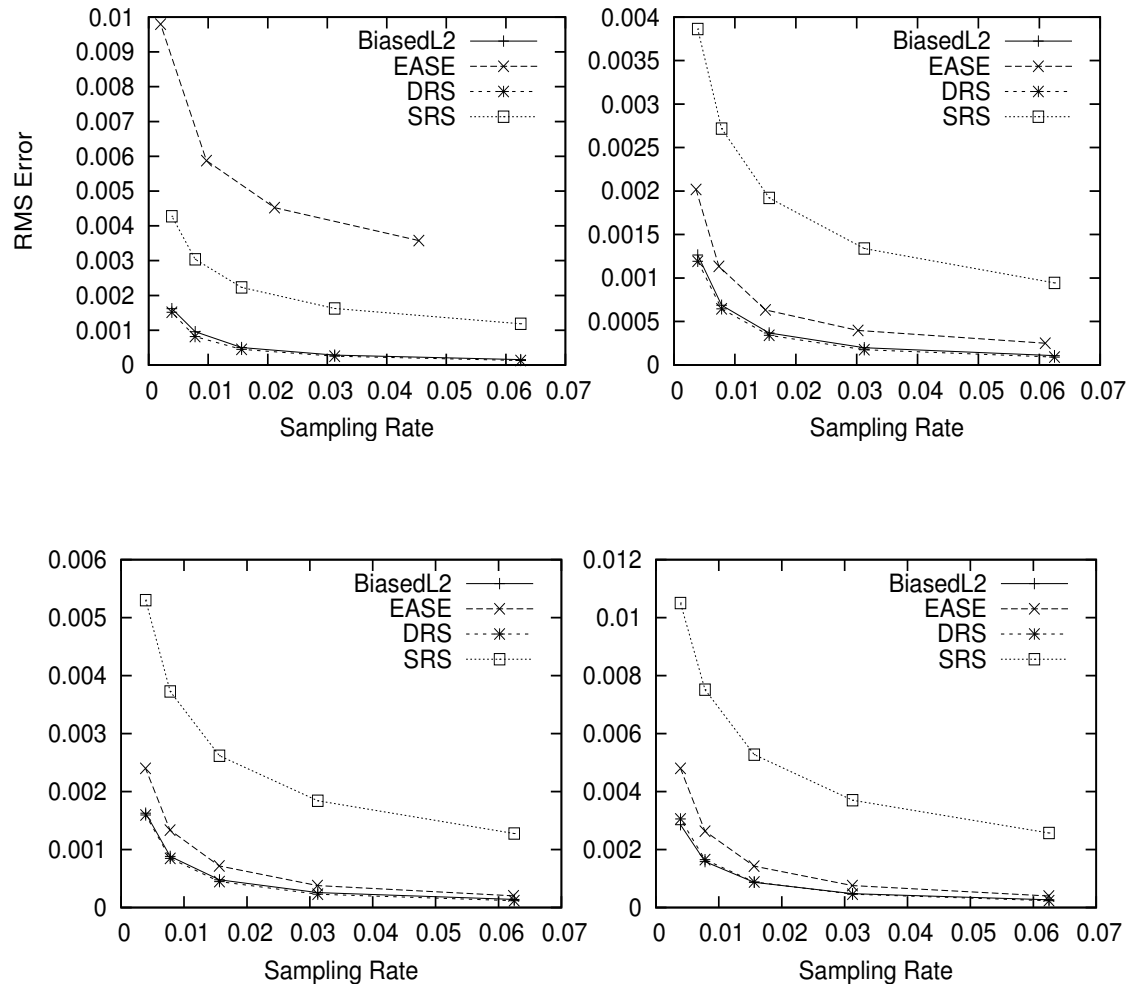


Figure 2.3: RMS error of SRS, EASE, Biased-L2, and DRS for four datasets (BMS1, T5, T10, T50).

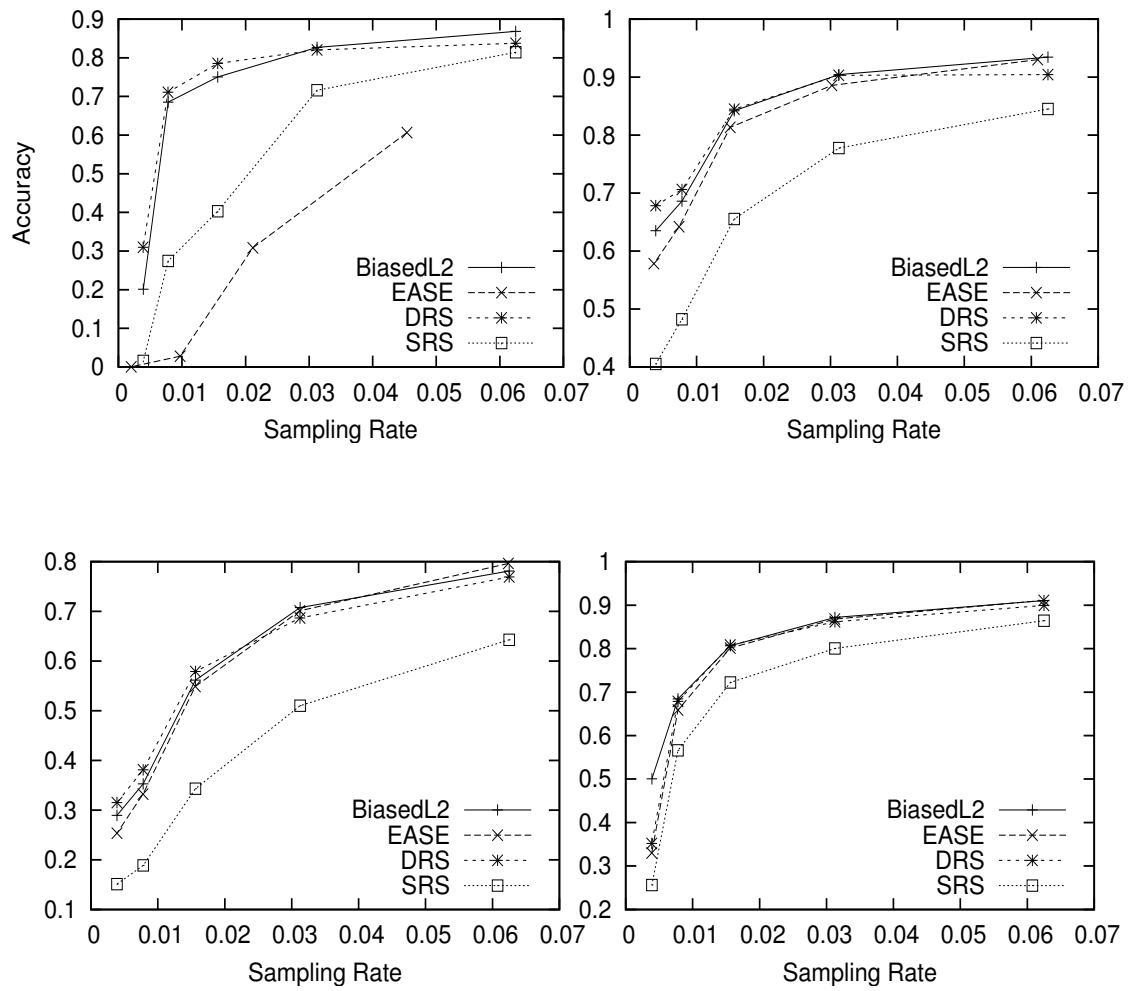


Figure 2.4: Accuracies of SRS, EASE, Biased-L2, and DRS for four datasets (BMS1, T5, T10, T50).

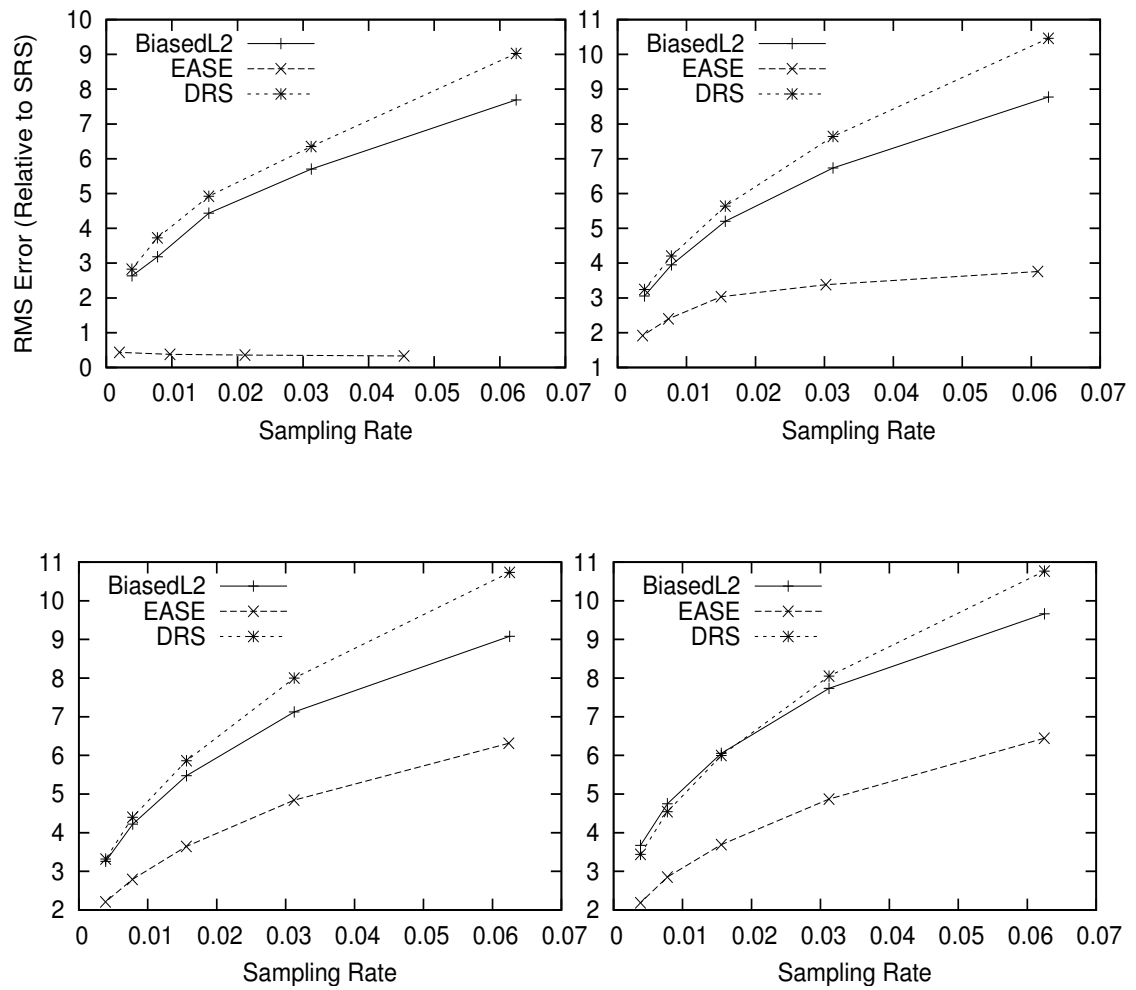


Figure 2.5: Ratio of the RMS error of SRS over EASE, Biased-L2, and DRS for four datasets (BMS1, T5, T10, T50). Higher  $y$ -coordinate values correspond to better quality samples.



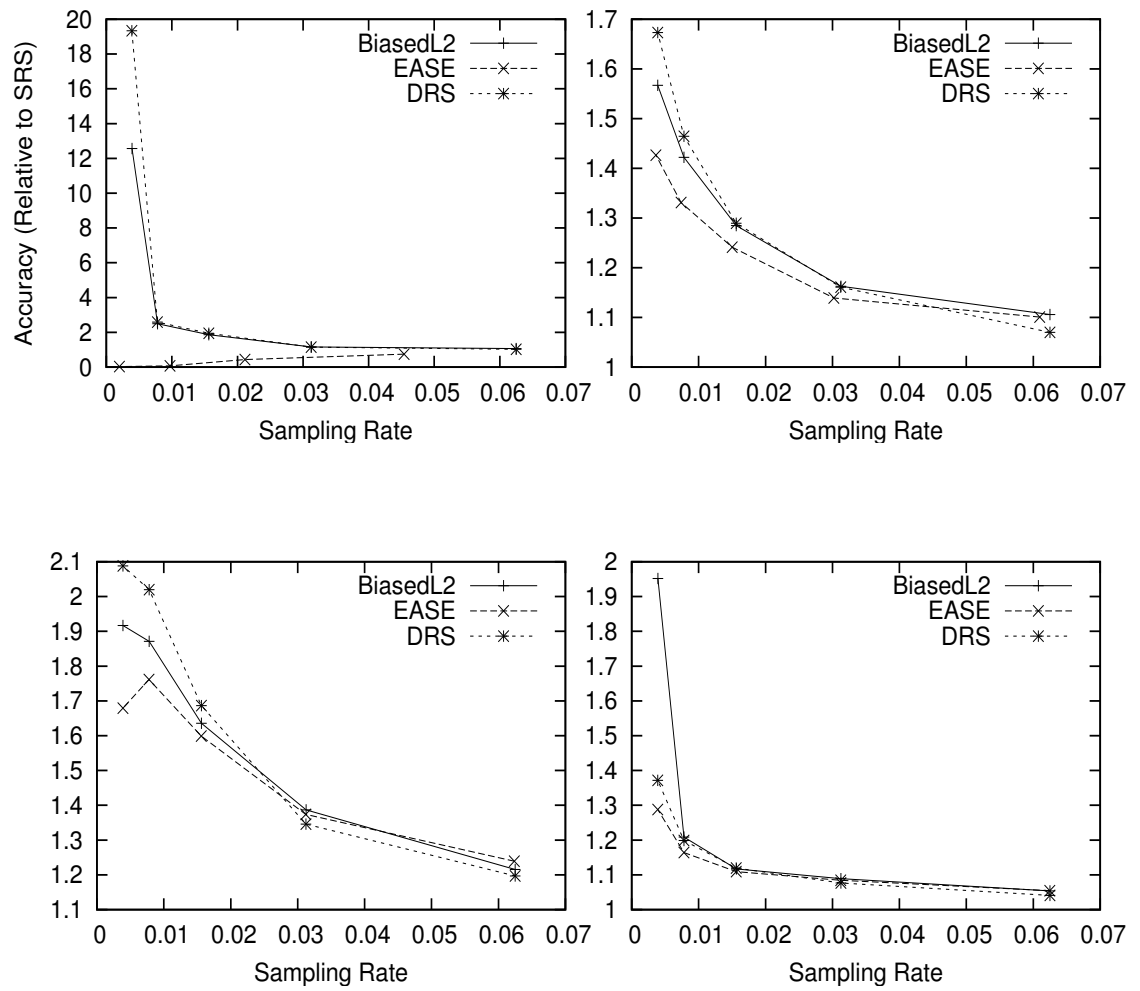


Figure 2.6: Ratio of the accuracy of EASE, Biased-L2, and DRS over SRS for four datasets (BMS1, T5, T10, T50). Higher  $y$ -coordinate values correspond to more accurate samples

Algorithm/Sampling Rate	0.062	0.03	0.015	0.007
EASE	1.03	1.10	1.13	1.11
Biased-EA	0.21	0.20	0.20	0.20
Biased-L2	0.18	0.20	0.19	0.19
DRS	52.2	21.1	9.6	4.9

Figure 2.7: Time spent per transactions (in milliseconds) for each algorithm, with various sampling rates.

On synthetic datasets the differences in accuracy results are slim, but DRS consistently more accurate than SRS. Looking at the accuracy results in Figure 2.4, we see that in some datasets, up to 90% accuracy is obtained by using only 3% of the dataset. This result is especially important when mining huge amounts of data. For applications where 90% accuracy is sufficient, instead of running the mining algorithms on the whole data, which can take even days for some datasets, a sample can be used, which is much smaller and easier to handle.

Although the EASE algorithm gives comparable accuracy bounds on synthetic datasets, it performs poorly on the real-world dataset (BMS1), both for sample quality and accuracy, see Figure 2.3–2.6 (leftmost). The result is not surprising since the real-world dataset is well known for this kind of behavior, such as, the owners of the dataset claim that most algorithms which work with synthetic datasets do not work well with this real-world dataset[48]. This is the main reason we selected this particular dataset as our real-world control dataset. We want to highlight the fact that DRS work perfectly both on synthetic and real-world datasets. On top of this, the major improvements of the present work over EASE are the running time and memory footprint of our new algorithm.

Finally, we compare CPU times for the algorithms we presented in this section. Figure 2.7 presents the average time spent in milliseconds to process one transaction for each algorithm on a Pentium IV 3Ghz computer. Biased-EA and Biased-L2 are up to

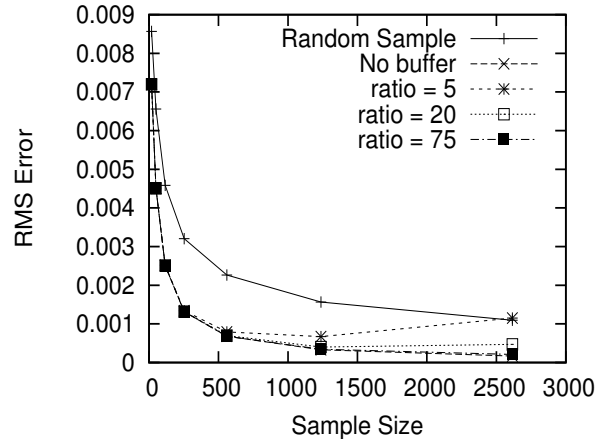


Figure 2.8: Effect of  $k$  on sample quality,  $ratio = |S_{DRS}|/k$

5 times faster compared to EASE. The main reason for this speed-up is the single pass structure of these algorithms compared to logarithmic halving steps in EASE. The CPU time of DRS varies with the sample size, as expected. More details about the running time of DRS is presented below in Section "Changing the update rate".

### 2.3.3 Extensions to DRS algorithm

In the previous section we presented the accuracy and quality results of the samples generated by the DRS algorithm. In this section, we further present the additional properties of the DRS algorithm; using  $k$  parameter to control the run-time performance and the fast-recovery property of the algorithm under distribution or sample size changes.

#### Changing the update rate:

In Figure 2.8, the effect of  $k$  on the sample quality of the DRS algorithm is presented. The figure plots the RMS errors of using different values of  $k$  on the BMS1 dataset. As the figure is plotted for different sample sizes, the results are given as the ratio of the sample size over  $k$ . We can see from the figure that the sample quality is similar to a random sample for bigger values of  $k$  (less updates), and quality increases for smaller values of  $k$  (frequent

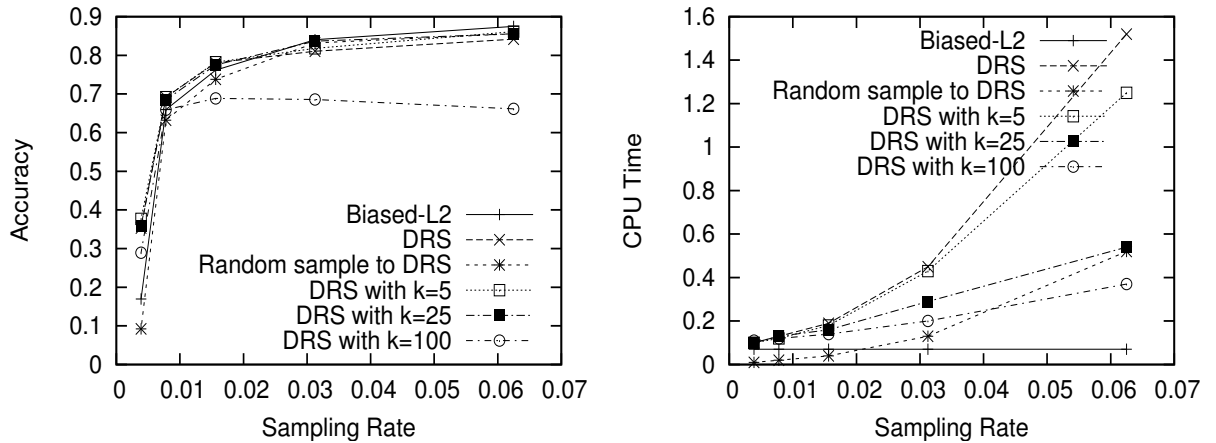


Figure 2.9: Accuracy and CPU time vs. sample rate for Biased-L2 and DRS with different  $k$  values (left, right).

updates). Figure 2.9 plots the accuracy result and the CPU time of the Biased-L2 and DRS algorithms for various sampling rates. The plots clearly show that the  $k$  parameter in DRS can be used effectively to control the trade-off between the running time of the algorithm and the quality/accuracy of the sample. For example, for sampling rate of 0.062, we can achieve up to a factor of 3 times speed-up in run-time by selecting  $k = 25$ , without sacrificing much from the accuracy. The effect of  $k$  on the runtime of DRS can also be seen in Figure 2.12. From this figure we can also observe that  $k = 25$  is a reasonable choice, since larger  $k$  values do not significantly decrease the algorithm runtime any further.

### Changing the sample size:

In the DRS algorithm we can change the sample size at any time of the sampling process, quite easily. When the sample size is increased from  $s_1$  to  $s_2$ ,  $s_1 < s_2$ , there occurs a gap in the sample of  $(s_2 - s_1)$  transactions. The next transactions from the dataset are added to the sample without any evaluation (in the most basic case). Similarly, when the sample size is decreased from  $s_1$  to  $s_3$ ,  $s_1 > s_3$ , we trim  $(s_1 - s_3)$  transactions out of the sample by finding the worst transaction in the sample and removing it without replacement, enough times. When adding or removing transactions, the item counts of the sample are updated

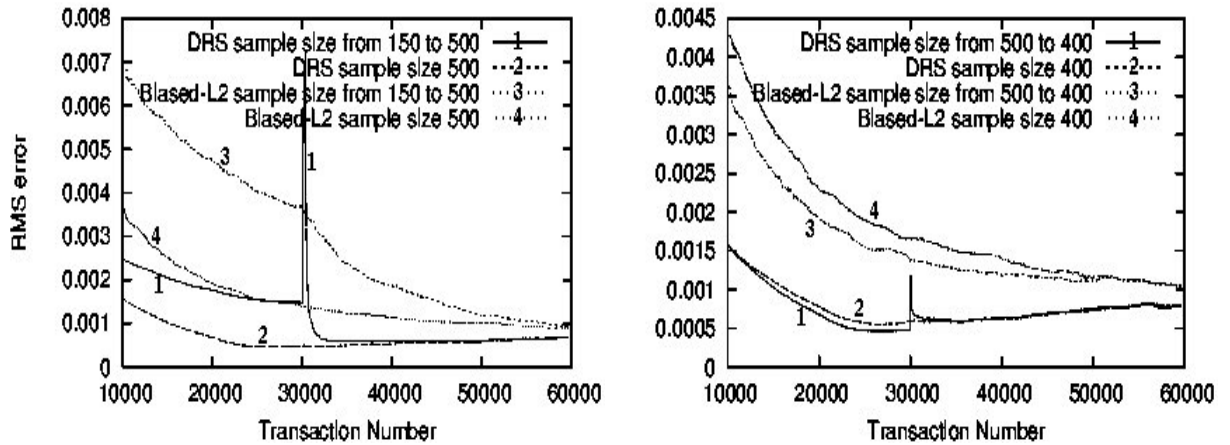


Figure 2.10: RMS error vs. number of elements processed, while (left) increasing and (right) decreasing the sample size suddenly after 30 000 transactions.

accordingly. After the sample reaches the desired size, it is used as the initial sample for DRS, and the sampling process resumes. Although theoretically it is hard to say how much changing the sample size on the fly affects the quality of the sample, empirical results show that this can cause a major increase in the cost function, but after a very short recovery period, the sample is as good as a deterministic sample again. In other words, once the DRS algorithm starts running again, the cost function decreases dramatically in a very short period of time.

The experiments in Figure 2.10 and Figure 2.11 are run on the real world dataset BMS1, while processing the transactions in the original order to prevent introducing free randomness in the dataset. In Figure 2.10 (left), the effect of increasing the sample size is presented. The lower line plots the trace of sampling the BMS1 dataset with a sample size of 500. The upper line plots the trace of sampling the same dataset with a sample size of 150 until the 30 000th transaction, after which the sampling rate is changed to target a final sample size of 500, which causes the jump in the RMS error. After a number of transactions, the RMS error function converges to the value it would get for a sample size of 500. One important point to note here is that, when adding new transactions to the sample we did not use any evaluation criteria, just to demonstrate the effect on the RMS

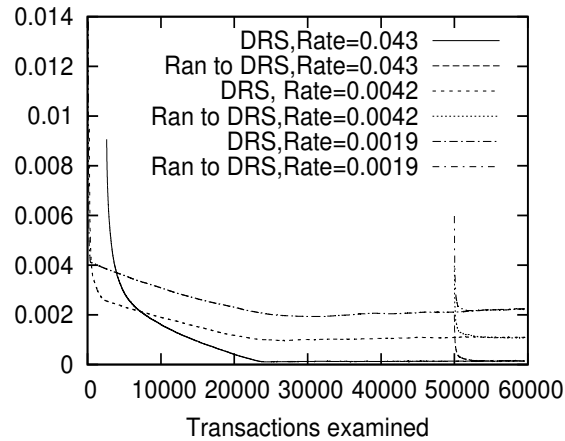


Figure 2.11: Plot of RMS error vs. number of transactions examined while converting random to deterministic samples for different sample sizes.

error value. One way to add more transactions is to make the whole process greedy, such that the peak caused by the sample size change would be lower, and the sample would converge gradually. Figure 2.10 (right) similarly shows the plot of decreasing the sample size from 500 to 400, and shows that the final RMS error value converges to the value it would get for a sample of size 400 (from the first transaction). These results show us that the DRS sample size can be changed at any time during sampling, and the jump in the error function can be compensated for with the RMS error converging to its normal value for the new sampling rate, after examining only a small number of transactions (typically proportional to the size of the sample). Note that the convergence for Biased-L2 is much slower, which clearly shows the better recovery of DRS after a sudden change in sample size.

Another important outcome of the fast convergence of DRS is that we can take a random sample from the dataset at any time, use this sample as initial sample for DRS, and convert this random sample to a new sample after only examining  $O(s)$  new transactions from the dataset, yielding the expected RMS error of a deterministic sample for that new sample size. Figure 2.11 shows the plot of three different sampling processes. The dataset BMS1 is sampled three times with sampling rates of 0.043, 0.0042, and 0.0019 (sample

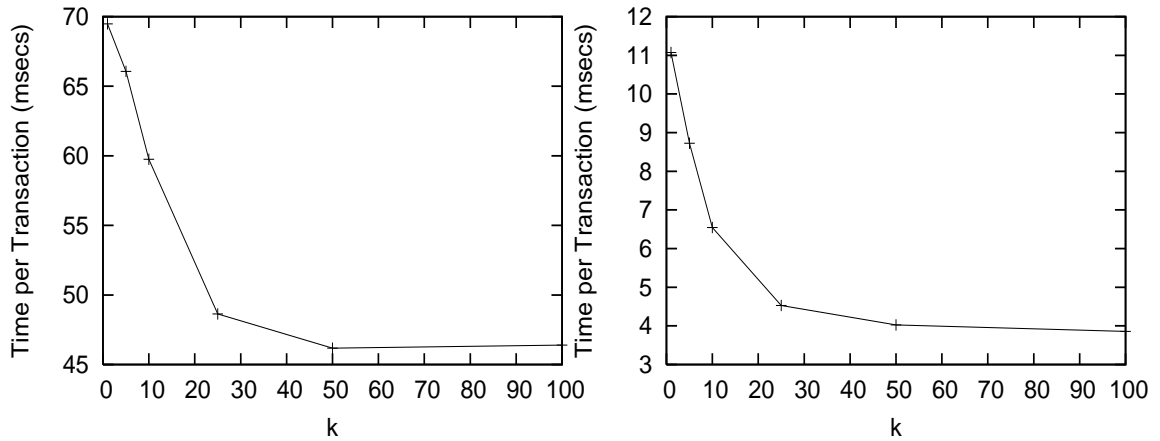


Figure 2.12: Effect of changing the  $k$  parameter on the speed of the DRS algorithm. Synthetic dataset (T10I6D100K) (left), and real-world dataset (BMS1) (right). For both datasets selecting  $k \approx 25$  seems quite reasonable. The time spend per transaction on each dataset varies with the average number of items per transaction. The synthetic dataset we test has more items per transaction on average than the real-world (BMS1) dataset, which increases the time per transaction on the synthetic dataset.

sizes of 2615, 253, and 116 respectively). Also, after examining 50 000 transactions, a simple random sample is created for each case, having exactly the same sizes as 2615, 253, and 116. The plots after transaction 50 000 show the results of using these random samples as initial samples for our algorithm. Clearly, after only examining a small number of new transactions, the RMS errors of the samples converge to the expected value. In the end, it makes little to no difference if we sample the whole dataset one transaction at a time, or if we get a random sample at any time and convert it using our DRS algorithm.

To sum up the experiments in this section, in terms of sample accuracy and quality, both Biased-L2 and DRS outperform EASE and SRS. Biased-L2 is our algorithm of choice if speed is important, and DRS is our choice if either the sample size must be chosen exactly, or if the sampling rate and/or data distribution changes frequently and fast convergence is needed.

## 2.4 Concluding remarks

In this part of the thesis we have presented DRS, a deterministic sampling algorithm. DRS is designed to sample count data, which is quite common for data mining applications, such as market basket data, web clickstream data and network data. Our new algorithm improves on previous algorithms both in run-time and memory footprint. Furthermore, by conducting extensive simulations on various synthetic and real-world datasets, we have shown that DRS algorithm generates samples with better accuracy and quality compared to the previous algorithms (SRS and EASE). For sudden changes in the distribution, DRS has the ability to remove under- or over-sampled transactions if a more suitable one is found during sampling. In the previous algorithms surveyed and in Biased-L2, transactions added in the early stages of sampling affect the overall quality of the sample especially if the distribution of the dataset changes; DRS is not subject to this limitation.



## Chapter 3

### Distributed Sampling Algorithms For Sensor Networks

We focus on sampling data from a set of sensor nodes linked by a network and consider the problem of extracting the sample from the network to a central DBMS and later querying it to find answers to conditions inside the sensor network. These queries may be based on snapshots, or may be continuous. The data collected by the sensors is usually highly redundant, and thus one need not collect and process all of it, approximate query results are usually sufficient. Hence, substantial savings may be obtained by either aggregating or processing the data in-network, or by obtaining a much smaller but representative sample which can then be processed by a centralized more powerful unit.

One particular kind of data we focus on is the multi-dimensional count data that arises fairly often from market basket data, census-like applications, or in the context of sensor networks, from monitoring several parameters of an environment. For instance, Mainwaring *et al.*[56] engineer Mica Weather Boards to record temperature, humidity, and barometric pressure. Additional components like photo-resistor and infrared thermopile sensors can join forces to infer other parameters such as cloud cover, altitude or wind speed. Levon[52] is developing a biosensor array for detecting biological warfare agents; this array can report and monitor the presence of a number of chemical compounds, and the data is inherently multi-dimensional as we are not only interested in each parameter but also in their interaction. In this context, one may desire to investigate possible correlations,

to discover that certain combination of values occur more often than others if certain conditions are met (association rules), or to construct a data cube (or iceberg cube if only the most significant statistical data is desired). Note that outliers are also of potential interest, especially to act as triggers; they can be monitored using completely different techniques, so we do not consider those here, rather we focus on extracting the most significant statistical trends.

We contend that collecting and maintaining a representative sample of the data from the sensor network is a promising data aggregation solution because samples, unlike other synopsis structures, are general-purpose and can be used as a surrogate for the (expensive) network[12]. In other domains, sampling has long been used to answer many aggregation queries approximately[33] such as MEDIAN, AVG, COUNT, and MODE. It also has been successfully used to speed up data mining tasks such as finding correlations and association rules[22, 16, 4, 66, 70], and iceberg cubes[58, 4]. In the context of sensor networks, using sampling as a surrogate can be most appropriate for this kind of global analysis, because the whole data must be available to find correlations, and gathering the whole data in a centralized unit requires prohibitive communication costs. Tailored aggregation procedures have better accuracy, but each requires its own communication costs, while the sample is computed and maintained once and enables any number of a wide range of queries out-of-network with still reasonable accuracy at no cost to the network. Moreover, an approximate answer often suffices and more precise results can be obtained otherwise by ‘drilling down’ in the network in an OLAP fashion.

Although random sampling is an easy and intuitive way to answer approximate queries, random deviations in the sampling process, however, make random sampling somewhat less precise and unpredictable. In the context of transactional data sets, a simple deterministic procedure (EASE [16], refined in Biased-L2 [4]) produces samples with errors consistently an order of magnitude better than that of a random sample. In the context of sensor networks, we find that this translates into order-of-magnitude communication and energy savings, namely, for an equivalent RMS error, our deterministic sample is much

smaller thus requires lower energy costs (including extra communication requirements of the algorithm). Alternatively, for the same resource spending, one gets a much more accurate picture.

We also would like to mention a few applications of our sampling scheme. By integrating spatial regions as categorical attributes, our sample will try to be representative in space as well as for the measurement. That is, it will have the same spatial distribution as the original set of sensors, at least at the resolution of regions. Note that the regions may overlap. The regions of a hierarchical subdivision such as quadtrees or hierarchical uniform grids may be desirable to ensure scale-independence, therefore regions adapted to a natural decomposition such as counties for census data may sometimes be more appropriate. In monitoring homeland security chemical threats, being able to detect the emergence and geographic spread of a hostile chemical compound[52] involves combining geographic information with that of the biosensor array to ensure that environmental conditions are accurately sampled. For this to be accurate, the sampling rates (both temporal and spatial) must be adjustable at the node level, which can easily be achieved by tweaking the weights in our weighted sampling approach.

The rest of this chapter is organized as follows: We give an overview of the previous work in Section 3.1. In Section 3.2, we present an arbitrary aggregation structure for sampling, discuss the motivation for weighted sampling in this kind of aggregation structures and finally present our deterministic sampling algorithm. In Section 3.3 we introduce the real world climate dataset we used in our simulations, and then experimentally study our algorithm by comparing to existing sampling algorithms, while focusing on issues such as sample quality, energy and communication costs. Finally, we conclude in Section 3.4 with both discussion and future work.

## Our Contributions

Our main contributions in this chapter are:

- We propose a novel deterministic weighted sampling algorithm as a new aggregation method for sensor network data. To the best of our knowledge, this is the first distributed deterministic sampling algorithm for sensor networks.
- Our algorithm weights the samples and adjusts the weights dynamically, which enables to work on networks organized in any arbitrary topology.
- In our deterministic weighted sampling algorithm, data aggregation via sampling is done on all (participating) nodes, which equally distributes sampling work over the network, and prevents any node from being a bottleneck (both CPU and communication based).

### 3.1 Related work

In this section, we give an overview of the previous work and state the relations and differences compared to our work.

In a preliminary version[5], we present the basic algorithm and experiments with the random dataset. In this version, we further extend on the previous work both in terms of the algorithm details and experiments.

In [4], Biased-L2 is presented, which improves on EASE[16, 15] to deterministically sample streaming count data. Both algorithms sample by introducing penalty functions using the support of each item. Each penalty function is minimized when the item frequency over the sample equals that over the whole data set, and increases sharply when item is under- or over-sampled. EASE reduces the number of records in the sample by applying halving steps on the sample, while Biased-L2 is a single pass algorithm that gives a final decision for adding the record to the sample or not by examining the record only

once. EASE and Biased-L2 are designed to work on centralized database settings, where data is stored in a database or comes from a single stream, so they are not directly applicable to sensor network settings, where data is distributed and network topology is unknown. We propose a novel deterministic sampling algorithm that uses similar ideas to Biased-L2, but extended to handle weights and distributed sources, essential for sensor network data extraction.

The research on stream processing is highly relevant to our work. Stream database systems aim to process data streams as fast as possible while avoiding the need to store the entire data. In most cases, this results in approximate answers to the queries. Aqua[33] uses an approximation engine to quickly answer typical aggregate queries while incrementally refining the result with the actual database results. Borealis[1] is a high-performance, low-latency distributed stream processing engine. In addition to typical snapshot queries, STREAM[46] queries in continuous fashion, which naturally adapts to stream data. Continuous queries are also used in sensor network databases TinyDB[55] and COUGAR[68].

Recently, data aggregation in sensor networks attracted great attention from the research community[54, 68, 38, 53]. In sensor networks, where the in-network processing of various aggregate queries is paramount, data aggregation inside the network could drastically reduce the communication cost (consequently, prolong the battery life) and ensure the desired bounds on the quality of data. Madden *et al.*[54] propose an effective aggregation tree (TAG-tree), which works on well defined epochs, and reduces the communication by using an optimum tree structure. We also use a tree generation algorithm similar to TAG-tree, where iterative broadcast from sink to the rest of the network is used to create a tree that covers the whole network. COUGAR[68] also creates aggregation trees similar to TAG. Sharaf *et al.*[63] propose more efficient ways of aggregation by exploiting group by queries, which works on top of TAG and COUGAR aggregation. In addition to tree aggregation, cluster-based aggregation approaches have also been investigated in LEACH[38, 39] and PEGASIS[53]. In LEACH[38], randomly selected cluster heads perform aggregation, and communicate directly with the base station to reduce energy use,

while in LEACH-C[39] the base station broadcasts the cluster head assignments, further improving the energy use of the network. PEGASIS[53] selects the cluster head by organizing nodes in a chain. Using only one cluster head at a time conserves energy at the expense of introducing latency issues.

Although the tree structure is optimal for communication, it is not robust enough for sensor networks. The robustness issue led researchers to propose a DAG architecture instead of an aggregation tree. Addressing the duplication problem of the DAG architecture, Considine *et al.*[25] and Nath *et al.*[59] independently propose using duplicate-insensitive sketches for data aggregation in sensor networks. While Considine *et al.* focus only on efficiently counting SUM aggregates, Nath *et al.* give a general framework of defining and identifying order and duplicate insensitive (ODI) synopsis, including a uniform random sample (DIR sample as we call it in this chapter). We compare our algorithm to DIR, which computes a uniform random sample of the sensor data on any arbitrary hierarchical network structure, using an optimum number of messages. The algorithm initially assigns random weights to data values. To generate a sample of size  $s$ , each node selects the sample values having the biggest  $s$  weights. The selected sample values are propagated up the hierarchy to the sink. Since during each sampling stage, the top  $s$  weighted samples are selected, after the final stage, a uniform sample of size  $s$  is obtained. Similar to this approach, our algorithm also uses a deterministic way to generate  $s$  samples for each node, thus effectively distributing sampling work equally over every node.

Manjhi *et al.*[57] by combining the tree and multi-path aggregation structures, propose a tributary-delta structure. Tree and multi-path structures coexist in this new structure, and convert to each other, so the tributary-delta can behave as effective as a tree and as robust as a multi-path, depending on the need and network topology. Shrivastava *et al.* propose an aggregation technique for medians in [64], specifically for sensor networks. Greenwald and Khanna [36] extend their space-efficient order statistics algorithms for sensor networks.

Compression is used recently as an aggregation method for sensor networks.

Lazaridis and Mehrotra[51] propose using a piecewise constant approximation algorithm to compress the time series with quality guarantees. Since the method proposed is lossy compression for single dimension time series only, applying it would not keep the valuable correlation information in multi-dimensional data. We assume our data is multi-dimensional (such as temperature, barometer, etc.) and we are mostly interested in the correlation of these dimensions. Deligiannakis *et al.* [29] propose extracting a base signal from the data and further using piecewise linear regression to compress the original signal, using the base signal. The algorithm explicitly uses the correlation between multiple dimensions of data (if exists) to increase the effect of compression. Multi-dimensional lossy compression methods (that keep correlations) and lossless compression methods are complementary to our work, since the samples we create are subsets of the original data, and any method to compress the data can be applied to our samples too. In this chapter, to focus on the original behavior of the algorithms, we present the algorithms without the added benefit of compression.

In their work, Heinzelman *et al.*[38] and Chen *et al.*[23] present energy formulas for wireless transmission and receive. According to [23], the energy usage during idle::receive::transmit is respectively 1::1.2::1.7. We also use the same energy formulas in our evaluations.

## 3.2 Distributed deterministic sampling

In this section, we first present the notation essential to understand our algorithm. Later, we discuss the shortcomings of the existing deterministic sampling algorithms for arbitrary sensor network topologies, and present the necessities for using weights in the sampling algorithm. Finally, we present our Deterministic Weighted Sampling (DWS) algorithm, which is designed to run on distributed environments. The simplicity and effectiveness of DWS is most appropriate to run on resource-restraint hardware such as sensor nodes.

### 3.2.1 Notation

Let  $D$  denote the database of interest,  $d = |D|$  the number of records,  $S$  a deterministic sample drawn from  $D$ , and  $s = |S|$  its number of records. Each record consists of a set of items. We denote by  $I$  the set of all items that appear in  $D$ , by  $m$  the total number of such items, and by  $\text{size}(t)$  the number of items appearing in a single record  $t \in D$ . We let  $T_{\text{avg}}$  denote the average number of items in a record, so that  $dT_{\text{avg}}$  denotes the total size of the database (as counted by a complete item per record enumeration).

For a set  $T$  of records and an itemset  $A \subseteq I$ , we let  $n(A; T)$  be the number of records in  $T$  that contain  $A$  and  $|T|$  the total number of records in  $T$ . Then the support of  $A$  in  $T$  is given by  $f(A; T) = n(A; T)/|T|$ . In particular,  $f(A; D) = n(A; D)/|D|$  and  $f(A; S) = n(A; S)/|S|$ . The distance between  $D$  and  $S$  with respect to the 1-itemset frequencies can be computed via the L2-norm (also called ‘root-mean-square,’ or RMS),

$$RMS(D, S) = \sqrt{\sum_{A \in I} (f(A, D) - f(A, S))^2} \quad (3.1)$$

Our goal is to select a sample  $S$  of  $D$  of size  $\alpha|D|$  (where  $0 < \alpha \leq 1$  is the *sampling rate*) that minimizes the distance  $RMS(D, S)$ . Although L2-norm is our choice of distance metric in this work, there are other possible notions of distance, for instance the *discrepancy* ( $L_\infty$ -norm) can be used if the maximum deviation is important.

Finally, to relate the notation to our sensor data, we assume that nodes have multiple sensors on them (such as temperature, barometer etc.). Each reading from a sensor corresponds to an item. A total reading from a node includes all the sensor values for a particular time. The collection of all the sensor readings correspond to records, and a whole record is transferred between nodes during aggregation.

### 3.2.2 Aggregation data structure

In sensor networks we assume that the data is distributed over the nodes, hence each node  $x$  holds a subset  $D_x$  of  $D$  such that  $\cup_x D_x = D$  (In the extreme case, each



node holds a single value, which may be updated over time). We also assume that an aggregation tree structure is already available for our algorithms. We do not require any specific property on the tree, other than its connectedness (it must cover all the nodes). Once the aggregation tree is built, sensor nodes, starting from the leaves of the tree, create a sample of size  $s = \alpha_x |D_x|$  from their data  $D_x$ , and forward these samples to their parents. Nodes in the middle levels of the tree wait until they gather samples from all their children (or for a timeout), and then do sampling on all the gathered data, including their own, to create a sample of size  $s$ . Since the desired sample size  $s$  is known, and  $D_x$  depends on the number of children, each node adjusts its sampling rate  $\alpha_x$  accordingly. This sampling scheme is similar to the DIR [59], and maintains a sample of size  $s$  for every node in the network.

### 3.2.3 Motivation for weighted sampling

One important challenge for us in distributed sampling is that, since we do not enforce any structure on the aggregation tree, the tree topology changes with the underlying sensor network topology, and each node in the tree has an arbitrary number of children. In particular, the sampling rate on each node varies depending on the number of children. Simply gathering all the children's samples in a big chunk of data and deterministically or randomly sampling over this chunk would introduce misrepresentations in the sample of a node. Namely, the sample values of nodes closer to the sink in the tree would have more chance to appear in the final sample. An example aggregation tree clearly showing this situation is presented in Figure 3.1. The top node has three samples of size  $s$ , the one coming from the left subtree, the right subtree and finally its own data. In this case, each sample has a different weight and a regular sampling by ignoring these weights would give each of these three samples a  $1/3$  chance to appear in the final sample. This means  $1/3$  chance for the top node's data,  $1/6$  chance for each left-subtree node's data and  $1/9$  chance for each right-subtree node's data. As we would like to give each node's data an equal  $1/6$  chance to be represented in the final sample, either the bottom nodes should send more

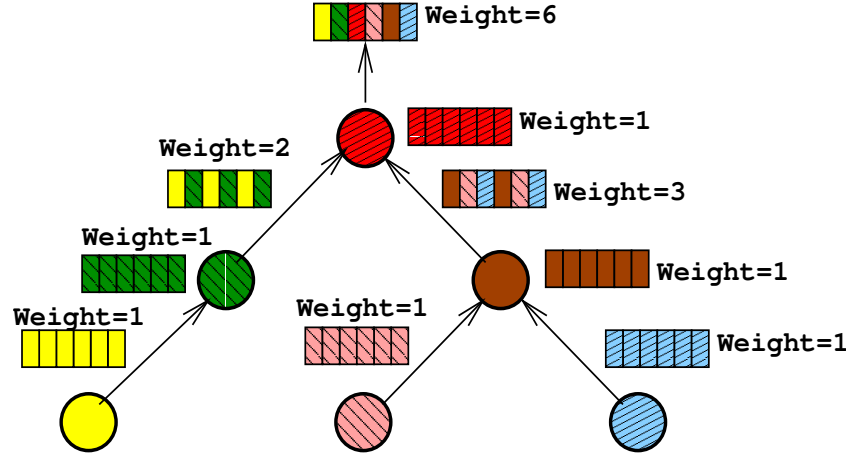


Figure 3.1: An example aggregation tree showing the nodes, samples and the weights for each sample. Each node gathers samples from its own data and children’s samples and creates a sample of size  $s$ . The sampling process is well distributed on each node, since each node maintains a sample of size  $s$ .

samples (vastly increasing the communication), or the samples should be weighted and the sampling algorithm has to be designed accordingly. This is the main reason existing centralized algorithms (EASE, Biased-L2) don’t work on sensor networks. To overcome this difficulty in our algorithm, we introduce weights for each sample, which are simply the representations of how many other nodes this sample stands for. Deterministically sampling the gathered data using the weights, we guarantee that each node’s data has the same chance to belong to the final sample, independent from its provenance in the network.

### 3.2.4 Deterministic Weighted Sampling (DWS)

We discussed the motivation for weighted sampling in distributed environments in the previous section. In this section, we present the main weighted sampling algorithm. DWS (pseudo-code given in Figure 3.2) handles weighted sampling at a node  $x$ , where the data comes locally from  $D_x$  with a weight of 1, and otherwise from the samples in its children  $y$ , each with a weight  $W_y$ . Thus  $W_x = 1 + \sum_y W_y$ , and these weights can

be accumulated in the network at a very low cost. Further, the weights are normalized by  $\alpha_x W_x$  as in lines 1–3, and we abuse the notation slightly to use  $w_j \leftarrow w_y$  for each record  $j$  coming from source  $y$ .

Based on the RMS distance function in Equation (3.1), but allowing weighted processing in addition, the algorithm uses the following penalty function,

for each item  $i$ :

$$Q_i = (r_i - \alpha n_i)^2,$$

where  $r_i$  and  $n_i$  are the weighted counts of item  $i$  in the sample and the dataset, respectively. The total penalty is  $Q = \sum_{i \in I} Q_i$ . For each  $i \in j$ , accepting a record  $j$  increases  $r_i$  by 1 and  $n_i$  by  $w_j$ , while rejecting a record only increases  $n_i$  by  $w_j$ , where  $w_j$  is the weight of the record  $j$  in dataset. During sampling, each record is added to the sample with a weight of 1. After sampling finishes the record weights are updated to reflect the total weight of that sample. Since the total weight of the sample is the number of other nodes it represents, the weight is simply the total number of contributing nodes in the subtree up to the node in question, which is easily accumulated at each sampling stage.

In order to accept a record into the sample, the penalty value should decrease, namely,

$$[(r_i + 1) - \alpha(n_i + w_j)]^2 - [r_i - \alpha(n_i + w_j)]^2 < 0,$$

Refactoring the following per item condition, we obtain:

$$1 + 2(r_i - \alpha n_i - \alpha w_j) < 0,$$

Summing over all items in  $j$ , we get

$$\text{size}(j) + 2 \sum_{i \in j} (r_i - \alpha n_i - \alpha w_j) < 0,$$

After further manipulation, the condition becomes

$$\text{size}(j) + 2 \sum_{i \in j} (r_i - \alpha n_i) - 2\alpha w_j \cdot \text{size}(j) < 0.$$

```

DWS( $D, W, x, \alpha_x$ )
1:  $w_x \leftarrow 1/(\alpha_x \cdot W_x)$ 
2: for each child  $y$  do
3:    $w_y \leftarrow W_y/(\alpha_x \cdot W_x)$ 
4: end for
5: for each item  $i$  do
6:    $n_i \leftarrow r_i \leftarrow 0$ 
7: end for
8: for each record  $j$  in  $D_x \cup (\cup_y D_y)$  do
9:    $sum_n \leftarrow 0, sum_r \leftarrow 0$ 
10:  for each item  $i$  in  $j$  do
11:     $sum_n \leftarrow sum_n + n_i; sum_r \leftarrow sum_r + r_i$ 
12:     $n_i \leftarrow n_i + w_j$ 
13:  end for
14:   $R \leftarrow sum_r - \alpha_x \cdot sum_n$ 
15:   $K \leftarrow 2 \cdot \alpha_x \cdot w_j - (2 \cdot R)/\text{size}(j)$ 
16:  if  $K > 1$  then
17:    Insert  $j$  into the sample  $S_x$ 
18:    for each item  $i$  in  $j$  do
19:       $r_i \leftarrow r_i + 1$ 
20:    end for
21:  end if
22: end for
23: return ( $S_x, W_x$ )

```

Figure 3.2: The Deterministic Weighted Sampling algorithm

The acceptance rule then becomes:

$$2\alpha w_j - \frac{2 \sum_{i \in j} (r_i - \alpha n_i)}{\text{size}(j)} > 1$$

which is the acceptance rule used in Figure 3.2 (lines 12–13).

Our method works by extracting from the sensor data a set of multi-dimensional boolean attributes. Categorical attributes are easily reduced to this setting by introducing a boolean characteristic attribute for each discrete category; numerical attributes can be reduced to it as well by associating a boolean attribute to a range in some shared histogram. Each boolean attribute is considered as a subset of the sensors (those for which there is a match). It strives to select a sample that simultaneously minimizes the frequency error over all the attributes.

As the communication costs are the dominating factor in sensor networks, we present the performance of the algorithm in number of messages. Assuming each sample value fits in a single message, the sample size of each node is  $s$ , and the total number of nodes is  $k$ , our algorithm performs a full sampling with  $O(k s)$  messages. The memory requirement of the algorithm is  $O(m + s T_{avg})$  per node, based on storing the counts for each item ( $m$  such items), weights for each record and the raw sample. Practically speaking, if we assume we are using a dataset with 5 tuples per record (similar to our real world dataset described further in Section 3.3.1), we need to use 6 integers (24 bytes) to store a record with its weight. Typically, keeping a sample of 1,000 records requires 24 Kbytes. The space required for the item counts is flexible and independent from the sample size. We can use a higher granularity for item counts to increase the accuracy of our sample, or if application permits, we can decrease the granularity to save memory for additional sample values. For example, if we use 100 equi-sized buckets per tuple to represent item counts, we need 2 Kbytes to store integer counts for the items. Since a typical sensor node (ex. Berkeley Motes) currently has 128 Kbytes of memory, we assume these are reasonable memory requirements.

### 3.3 Experiments

In this section, we now evaluate our DWS algorithm both on real and synthetic data, and compare with the DIR sampling algorithm[59] based on sample accuracy and energy usage. We demonstrate our work using the wireless sensor network simulator Shawn [49]. Shawn is a discrete event simulator that allows us to test our algorithms on more nodes than the other well known simulators.

We introduce the real world climate dataset in Section 3.3.1. In Section 3.3.2 we give the simulation results of our deterministic algorithm (DWS), and compare the results to other algorithms such as naive random sampling and Duplicate Insensitive Random (DIR) sampling. We show that our algorithm has many advantages such as better sample quality and less communication rate compared to other methods. Finally, Section 3.3.3 demonstrates the performance of our algorithm in answering typical SQL queries such as AVG, COUNT, SUM, MAX, etc. over climate data set.

All energy usage calculations in the experiments are based on total number of sent and received messages. Using the weights from[23], the received messages are weighted with 1.2, the sent messages are weighted with 1.7 and energy usage for a node is the total. Each tree building message is calculated as a single message. When exchanging samples between nodes, each record is assumed to fit in exactly one message, also the weights of the whole sample are assumed to fit in one message. These are reasonable assumptions, since there is only limited number of sensors on each node, so the nodes can transfer all readings in one single message.

#### 3.3.1 Climate dataset

In our experiments, we used real world climate data from U.S. National Climatic Data Center<sup>1</sup>. The dataset is the daily readings of precipitation, snow fall, snow amount, maximum and minimum temperature values throughout the continental U.S., cov-

---

<sup>1</sup><http://www.ncdc.noaa.gov/oa/climate/research/ushcn/daily.html>

ering 1064 stations for nearly the last 100 years. Since there are gaps in the station readings, we selected the maximum possible stations covering the exact same days. We gathered 227 stations having 4153 daily readings per station. The time period of the readings is presented in Figure 3.4. The black marked regions show the available days while the white regions show the gaps between daily readings. The real locations of the 227 stations on the U.S. map<sup>2</sup> is also presented in Figure 3.5.

For our real data experiments, we simulated the sensors on the U.S. map (scaled) by associating each sensor with a station, and positioning 227 sensor nodes on a 60x25 region, according to real station positions (latitude and longitude) on the map. The sensor node corresponding to the station in Central Park, NY is selected as the sink. To preserve the time and location correlation of data, we further fed each sensor with the exact data collected from the associated station, in original order. This way, each reading of a sensor corresponds to a daily data of the associated station. The average values of the 5 tuple data are plotted in Figure 3.3.

Analyzing the climate data, we also realized that there exist many inconsistencies within the data, because of measurement equipment failures in early days, and metric conversion errors. We left these inconsistencies as is, since they present additional challenge for us in the dataset, and these kind of perturbations in data are quite likely in real world sensor deployments.

### 3.3.2 Distributed data reduction

In this section, we give the simulation results of our Distributed Weighted Sampling algorithm on sensor networks. The evaluation is based on two categories, the sample quality and the energy usage. In order to demonstrate our work, we also include the naive random sampling algorithm and a more advanced, Duplicate Insensitive Random (DIR) sampling algorithm from Nath *et al.*[59].

Two different datasets are used in the simulations, the climate dataset we intro-

---

<sup>2</sup>Generated using tools from <http://www.planiglobe.com>

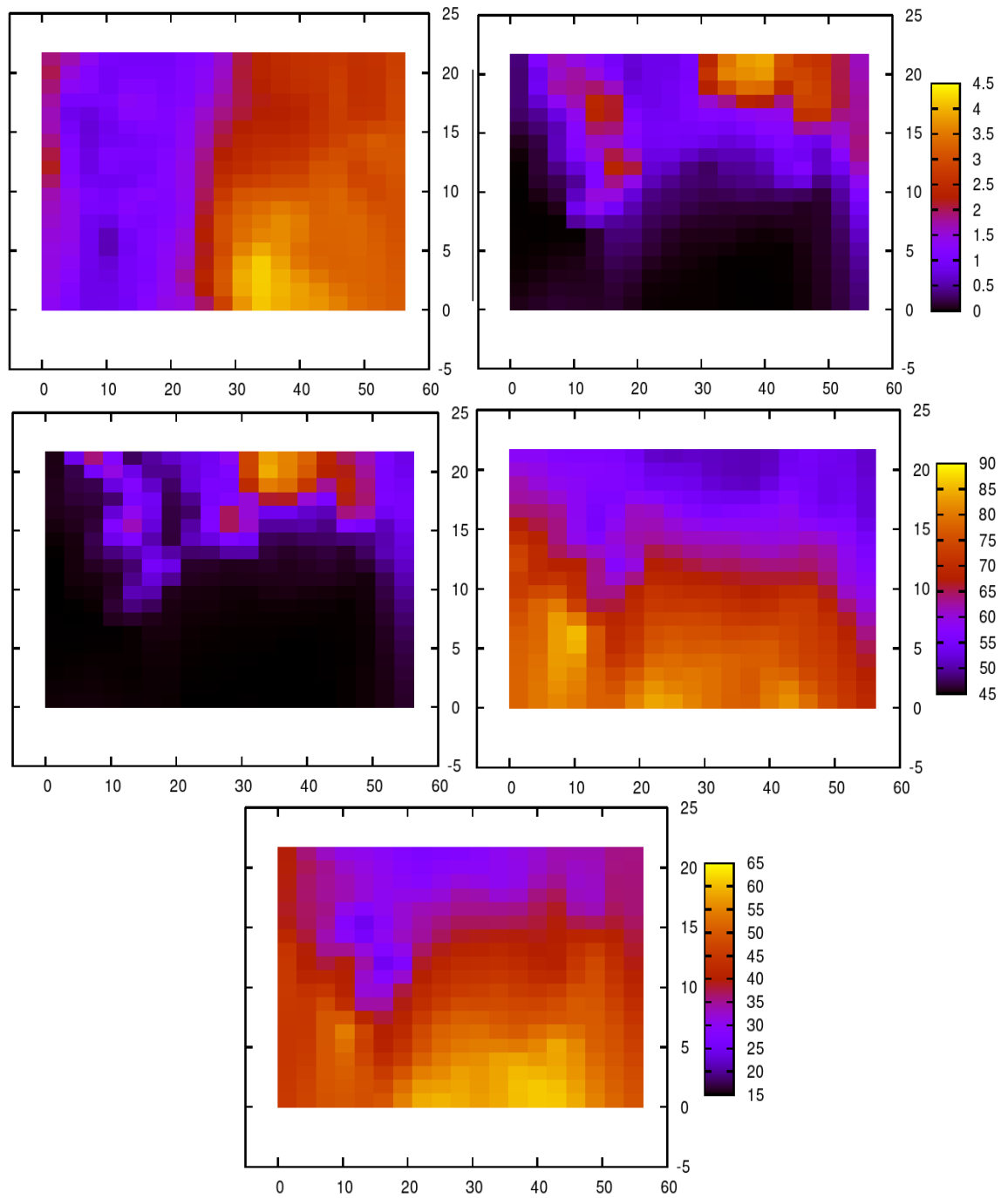


Figure 3.3: Average values of precipitation (top left), snow fall (top right), snow amount (center left), maximum temperature (center right), and minimum temperature (bottom).



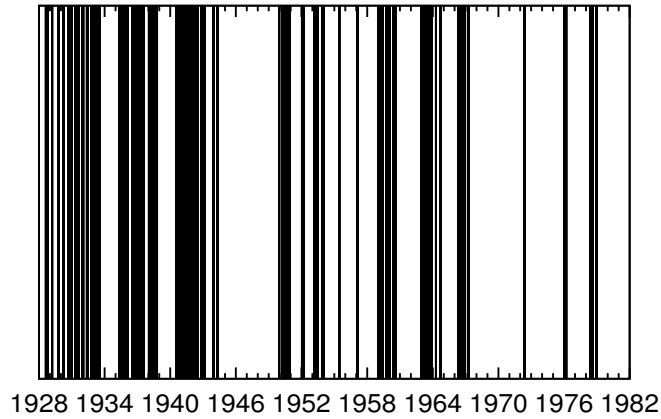


Figure 3.4: Time period for weather data used in the simulations.



Figure 3.5: Simulated U.S. area (60x25), and 227 sensor locations.

duced in Section 3.3.1, as the real world dataset and the synthetic dataset (T10I6D1500K), which is an association rule dataset, created using the IBM data generator [65]. The synthetic dataset includes 1, 500, 000 records, each having different number of items. The average number of items in each record is 10, the average number of itemsets is 6, and the total number of unique items is 1, 000. Items represent different sensor readings, and a record represents a total reading from a sensor, with various items. Items represent discretized readings from sensors; for instance, if we assume we have 10 sensors on a single node, and

each sensor reading is discretized into 100 buckets (such as temperature), a record having items 30, 140, 550 means the readings for the first sensor is 30, second one is 40, fifth one is 50 and there are no readings present for the rest of the seven sensors.

For the simulations with the climate data, we used 227 nodes, each having a constant radio range of 6 to ensure connectivity. The sink is placed on a predefined location as discussed in Section 3.3.1. For synthetic dataset simulations, we again used 227 nodes, randomly deployed in a 60x25 area, sink on the center.

Initially, for each algorithm, a data aggregation tree is created using the same algorithm as in [54]. In naive random sampling, each node samples  $s$  records and forwards the samples up the tree without any further data reduction. At the end the sink has  $sk$  records from a network of  $k$  nodes, which it further samples to generate a sample of size  $s$ . The details of DIR algorithm are already discussed in Section 3.1, and our DWS algorithm in Section 3.2.

The simulation results show the averages of running DWS algorithm on synthetic dataset and random algorithms (naive and DIR) on both datasets 100 times. Since climate dataset is a static dataset, and DWS is a deterministic algorithm, DWS is ran once on this dataset.

Figure 3.6 (a) and Figure 3.7 (a) show the RMS error values of the final sample generated by the algorithms for four different sample sizes 1000, 2000, 3000 and 4000 on both datasets. From these figures we can see that our DWS algorithm generates up to a factor of 4 times better quality samples than the other two algorithms, for both datasets.

Figure 3.6 (b) and Figure 3.7 (b) show the total energy used in the network while generating the samples. The energy usage is calculated based on the total number of messages sent and received as described in Section 3.3. The energy usage results are scaled for a clear presentation. As expected, the energy usage of DIR and DWS are the same, and up to a factor of 20 times less than that of the naive random sampling.

To clearly compare DIR and DWS algorithms, in Figure 3.6 (c) and Figure 3.7 (c), we generate two samples having the same quality based on RMS error values and compare

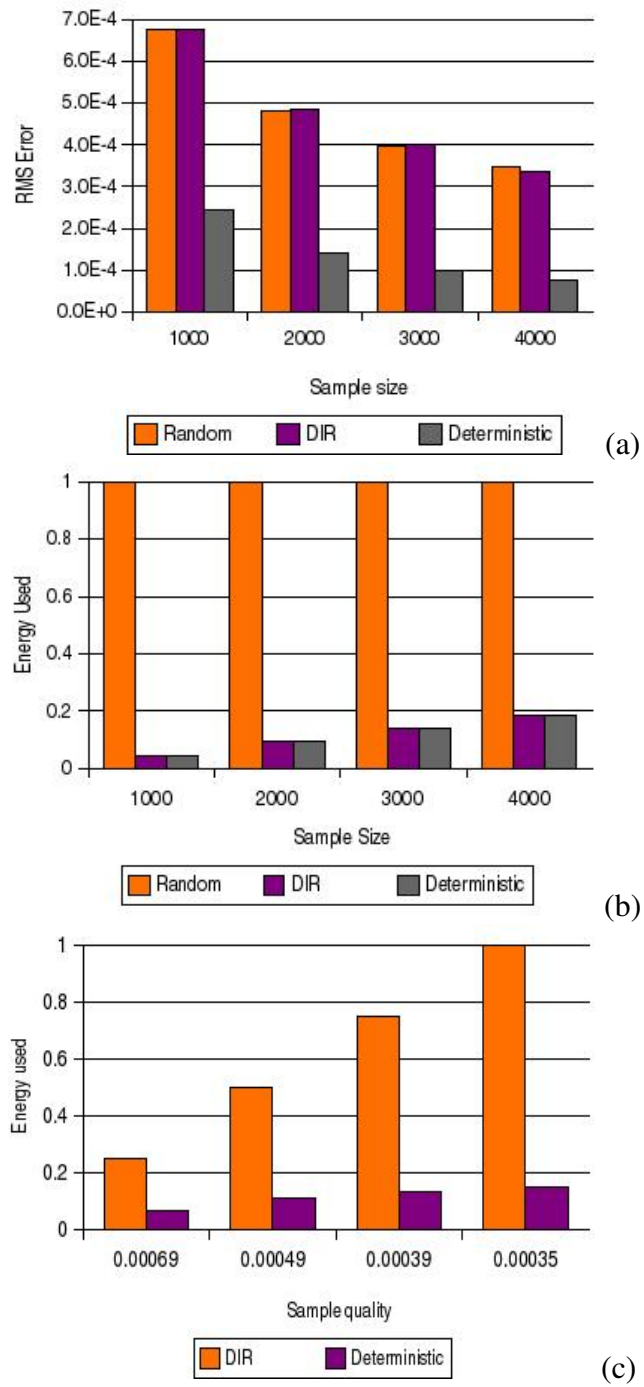


Figure 3.6: Results for real world climate dataset. (a) RMS results vs. sample size, (b) energy usage vs. sample size, and (c) energy usage vs. sample quality.

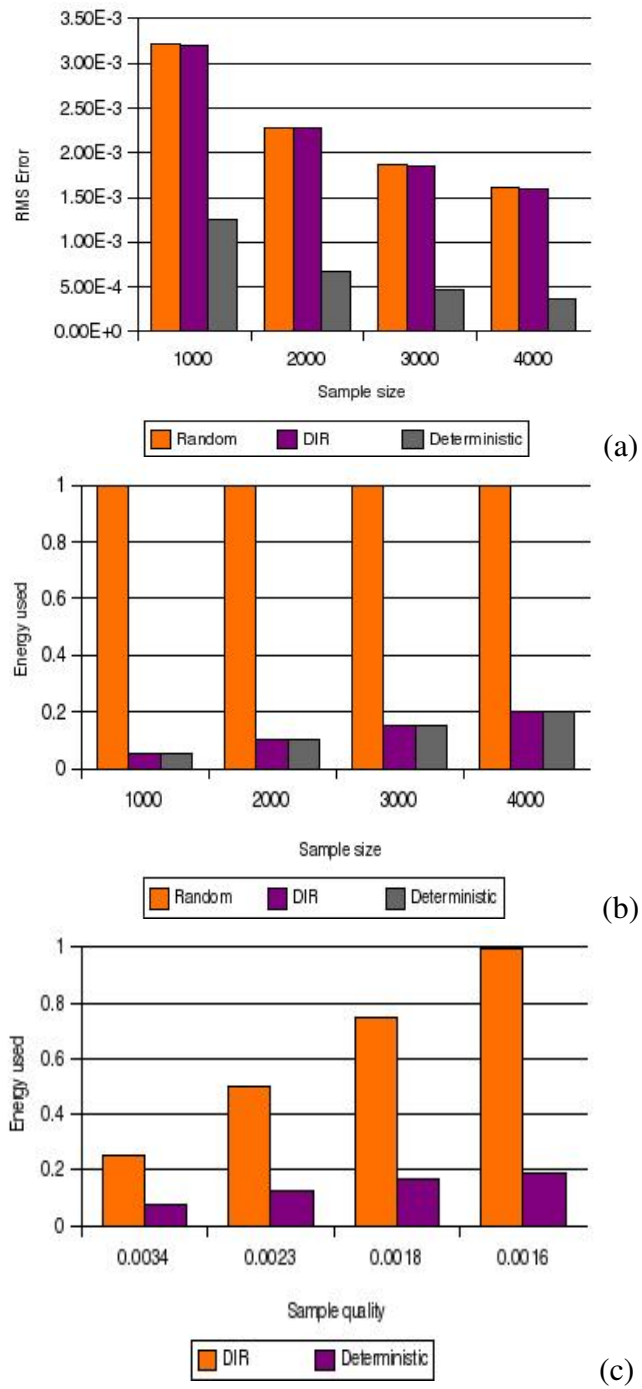


Figure 3.7: Results for synthetic dataset. (a) RMS results vs. sample size, (b) energy usage vs. sample size, and (c) energy usage vs. sample quality.

the energy used by both algorithms. Here, DIR algorithm still uses samples of sizes 1000, 2000, 3000 and 4000. The sample sizes of DWS algorithm are 268, 448, 532, and 600 for climate dataset, and 314, 505, 685, and 766 for synthetic dataset. For both datasets, DIR algorithm uses up to a factor of 6 times more energy than DWS algorithm. From the figures it is clear that, we can have the same quality sample by using considerably less energy if we use our DWS algorithm.

### 3.3.3 Example SQL queries

In this section, we evaluate answering SQL queries using the sample instead of the dataset after the sample is extracted and stored in a DBMS outside of the network. We presented sample quality results of our algorithm in the previous section. Those results were based on RMS distance between the dataset and the sample. Now we compare the two algorithms using AVG, MAX, SUM and COUNT queries. To achieve this goal, we use the real-world climate dataset as our dataset of choice. As we keep the original order of the records in this dataset, we created a single sample of size 4000 (sampling rate of 0.0036) using our deterministic algorithm, and 10 samples of size 4000 using the DIR sampling algorithm (as this is a randomized algorithm). Our deterministic algorithm on this static data always creates the same sample, so it is not necessary to generate multiple samples. On the other hand, the DIR algorithm is run multiple times to observe the average behavior of the algorithm, since the samples generated are different for each run of the algorithm. When presenting the results for the DIR algorithm, we present the min, max, mean and standard deviation of the errors over the 10 samples.

Samples are created once and all 4 queries are answered using these samples. The evaluation metric is the error rate between the query results of the dataset and the sample, calculated as:

$$ErrorRate = \left| \frac{Result_{dataset} - Result_{sample}}{Result_{dataset}} \right|$$

In Figure 3.8, the horizontal lines are the DWS error rates, which is a single value per

sample. The vertical lines are the min-max values for DIR error rate, and the boxes show the mean and the standard deviation values of the DIR error. Table 3.1 presents in detail the comparison of the DWS and DIR algorithms. DWS errors are compared with the mean, min and max errors of the DIR samples, and the results are presented as a ratio of  $\frac{DIR_{error}}{DWS_{error}}$  for easy comparison.

### Query 1

```
SELECT AVG(prcp), AVG(snow),
       AVG(snwd), AVG(tmax), AVG(tmin)
FROM dataset
```

This is a query to test the effects of sampling algorithms on AVG queries. The query selects the average values for all 5 tuples on both dataset and each sample, and Figure 3.8 (top left) and Table 3.1 show the error rates of both algorithms. Looking at the average error rates, except snow fall (snow), DWS algorithm outperforms DIR, sometimes by a factor of 24 times. Additionally, DWS error rate is much smaller than the worst case DIR samples (sometimes by a factor of 59 times), and close to the minimum error range of random samples. We would like to state here that it is unlikely for a single random sample to accurately answer all queries, the minimum error values for DIR are coming from different random samples, however we use only one DWS sample and being close to the minimum bounds of DIR samples shows how accurate our deterministic sample is.

### Query 2

```
SELECT COUNT(*)
FROM dataset
WHERE snow=0 AND snwd > 0
```

The second query finds the number of days when the snow fall amount is zero but there exist snow on the ground. Our focus is to test COUNT type queries. We are also testing if the correlation information between snow fall (snow) and snow amount on ground (snwd) are

preserved accurately in the samples. Figure 3.8 (top right) and Table 3.1 show the results for this query. As we claimed, the deterministic algorithm is more accurate to represent the correlation information in multi-dimensional count data. The deterministic sample is a factor of 8 times better than the average, 1.3 times better than the best and 18 times better than the worst random sample. Deterministic sample is the clear choice in this query as it outperforms random samples in all aspects.

### Query 3

```
SELECT MAX(prcp)
FROM dataset
WHERE tmax > 80
```

This is an outlier query testing the MAX value and also the correlation information between precipitation (prcp) and maximum temperature (tmax). The result is presented in Figure 3.8 (bottom left) and Table 3.1. Both algorithms are quite inaccurate finding the MAX value. Note that this result is typical, samples in general are not good for handling outliers. Detecting and handling outliers is another research topic[12, 33], and requires more specific data structures and techniques. We also would like to highlight that each of these specific techniques solve a specific kind of outlier problem, and there is no "one size fits all solution". Therefore all these techniques can be used additional to the sampling approach to generate a more complete data reduction solution. For this reasons, we only would like to demonstrate the shortcomings of sampling in general, and focus on finding samples for more general use.

### Query 4

```
SELECT SUM(snow)
FROM dataset
WHERE tmin < 0
```

The last one is to test SUM type of queries. Selects the total snow fall amount for days

DIR/DWS	Query-1					Query-2	Query-3	Query-4
	PRCP	SNOW	SNWD	TMAX	TMIN			
MEAN	2.50	0.86	1.31	5.33	24.41	8.39	1.00	5.47
MIN	0.33	0.26	0.12	0.21	3.56	1.35	0.90	0.77
MAX	7.20	1.60	4.13	9.74	59.26	18.17	1.06	10.85

Table 3.1: Ratio of the mean, min and max error values of the DIR samples to the error values of DWS sample for each query.

having minimum temperature less than 0 degrees. The results are in Figure 3.8 (bottom right) and Table 3.1. We can clearly see that, for this query, the deterministic sample is slightly worse than the best random sample, and a factor of 5 times better than the average and also a factor of 10 times better than the worst random sample. In this query also, deterministic sample is much more accurate in keeping the correlation information between snow fall and minimum temperature.

We tested our algorithms on real world dataset, using typical SQL queries. As expected the sampling algorithms give accurate results for SUM, COUNT, AVG type of queries, and quite inaccurate results when it comes to outlier queries such as MAX (see Figure 3.8 and Table 3.1). We also observe that deterministic algorithm is much more accurate in keeping correlations between items. The accuracy of the aggregation algorithm relates to the total energy usage of the network (communication to extract the sample), such as a smaller deterministic sample is more preferable to a bigger random sample, when both have the same accuracy. The results in Figure 3.8 and Table 3.1 also demonstrate that once a sample is extracted from the sensor network, it can be used to answer different types of queries (except outliers of course), with reasonable accuracy.



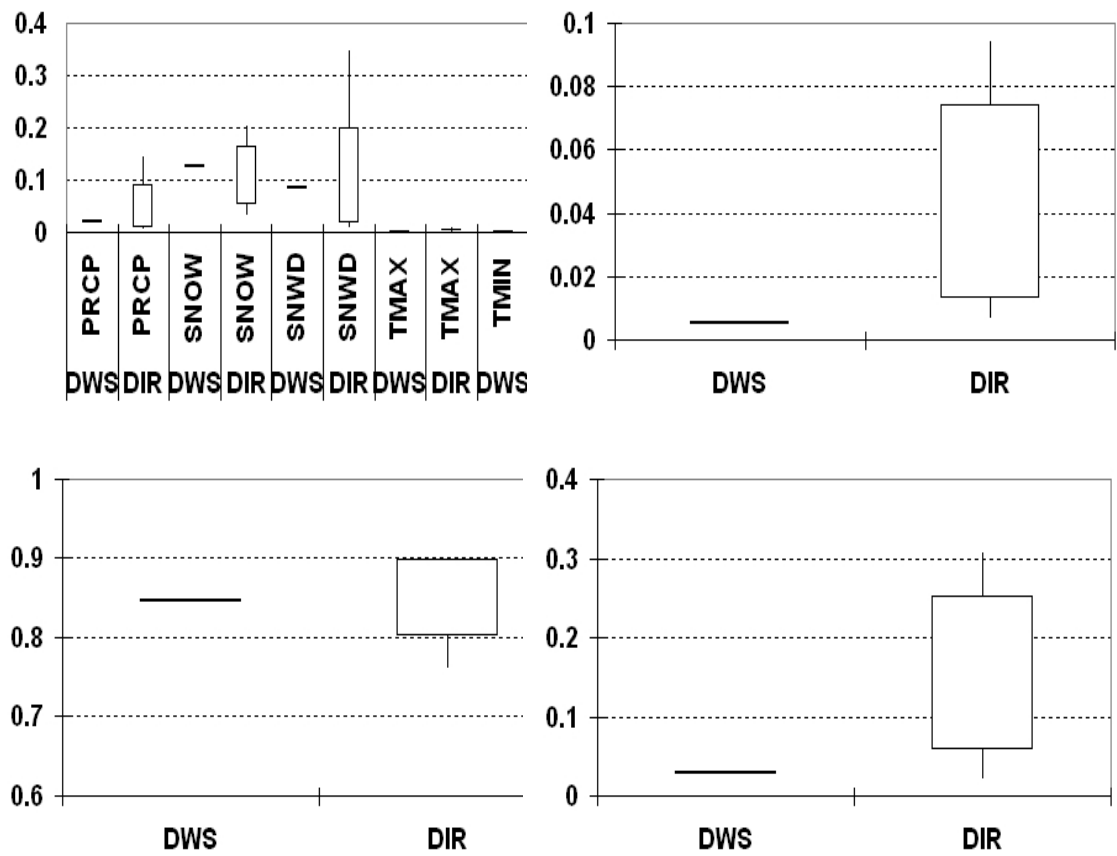


Figure 3.8: SQL query errors for deterministic sample and DIR samples. Results for Query-1 (top left), Query-2 (top right), Query-3 (bottom left) and Query-4 (bottom right).

### 3.4 Concluding remarks

We have presented DWS, a novel deterministic weighted sampling algorithm as a new aggregation method for network of wireless sensors. Deterministic Weighted Sampling is simple enough to not consume too many resources locally, and we validate through experiments that the sample it provides is vastly superior to other distributed sampling methods exist for sensor networks. Our algorithm is designed to work on arbitrary network topologies, by introducing weights for samples and dynamically updating these weights throughout the sampling. DWS by design effectively distributes the aggregation work over all the nodes by enabling each node to generate a fixed sized sample and prevents any node from being a bottleneck.

One criticism of our approach is that loss of connection in the aggregation tree structure induced by link or node failure can have drastic effects on the representativity of our sample, since an entire subtree may no longer be contributing to the sample. This can be handled by allowing a multi-path aggregation structure [59, 57]. In the context of aggregation, however, one must then address the problem posed by the duplication of the data along these multi-paths. The duplicate-insensitive solutions provided by Nath *et al.* [59] and Considine *et al.* [25] do not extend to our deterministic algorithm, and we leave it as a matter for future research to handle robust connectivity with our deterministic sampling algorithm.

Although our original design requires input from every sensor node, this can easily be remedied by incorporating other sparse sampling approaches. We note that our approach only concerns how the samples are aggregated along the network. For instance, the contributing nodes can be selected by another sampling approach such as the approximate uniform random sampling of Bash *et al.*[13]. Likewise, the temporal frequency at which measurements are updated at a node can be regulated by an adaptive process such as proposed by Jain and Chang[44]. In any case, known compression methods can be applied to the in-network sample aggregation to further minimize communication costs[28, 29, 51].

## Chapter 4

# GPS & Compass Free Node Localization On Wireless Sensor Networks

Wireless sensor networks are composed of hundreds, possibly thousands, of low-cost *sensor nodes* that are capable of making environmental measurements, performing computations, and communicating with one another. Most importantly, through small motors or motion actuators, these devices are capable of physically organizing themselves in order to cooperatively achieve a desired task [71].

An important problem in mobile sensor networks is each sensor's awareness of its position and direction of movement relative to the entire network. This problem is commonly known as *localization* [41]. In general, such location awareness empowers routing algorithms to determine the most efficient message paths [47], or to achieve goals such as optimal area coverage [61]. For example, in aggregation networks [26], node localization is needed in order to construct topology-aware routing overlays that will reduce message transmission time, increase reliability, and reduce power consumption. In routing applications, it is sufficient for nodes to be aware of the coordinates of their neighbors *relative* to a local coordinate system common to the entire network [20]. We call this *relative* localization since each node's position is relative to the local coordinate system. To support mobility applications, a node must move in a specific direction in a manner that is related to its neighbors. To achieve this, in addition to knowing its neighbors' positions relative

to a common coordinate axis, a node must be aware of its neighbors' positions relative to its own direction of movement. This is the node's orientation. We call *directional* localization the problem of determining the position *and* the orientation of each sensor in the common coordinate system. In the remainder of the chapter we may refer to “directional localization” as simply “localization,” unless otherwise noted.

Providing support for directional localization in mobile sensor networks is a difficult task. Traditional solutions rely on information supplied externally in one of two forms: (1) via global positioning systems (GPS) – which requires additional hardware at additional costs, or (2) via fixed-point reference nodes, or anchors, whose global locations are known a-priori [24]. Such methods are most commonly used in static networks [62]. Recent efforts on mobile networks assume that only a small subset of the moving nodes (seeds) use GPS [43].

Many applications require sensor network mobility in environments where GPS signals may not be available and pre-existing infrastructures do not exist. Consider a fire search mission inside a building where a set of mobile nodes explore a floor with the goal to locate the source of fire. The nodes move collaboratively, in a semi-rigid swarm, taking temperature measurements while following a path that covers the area. Additional factors may exacerbate the problem further. Environmental errors must be taken into consideration, otherwise due to node mobility the additive error in the estimated location can accrue, rendering any algorithm impractical. This is a consequence of mechanical errors in evaluating the direction and distance of movement, which may occur in-between individual measurements. This type of errors can be due to manufacturing defects or fluctuations in the environment (e.g. surface friction). Thus, as the movement of the network evolves, the uncertainty of the position and direction of a node increases.

Our main contributions are algorithms for solving the problem of *directional* localization in sensor networks with mobile GPS-free nodes. We introduce novel, motion-based algorithms for node position and direction calculation with respect to each individual node's local coordinate system in mobile ad-hoc sensor networks, without global position-

ing information. Our first algorithm, GPS-free Directed Localization (GDL), assumes the availability of a digital compass on each node, and calculates a node's directional localization from a single-step movement. In our second algorithm, GPS and Compass free Directed Localization (GCDL), we relax the compass requirement and compute directional localization with a 2-step motion algorithm. Our algorithms perform localization in a few steps of movement and have a small memory footprint; in addition, they are not affected by cumulative position errors. More specifically, our proposed algorithms:

- provide *directional* neighbor localization in a network-wide coordinate system,
- work under fairly large motion and distance measurement errors,
- are unaffected by the speed of nodes,
- support a stable network in mobility problems.

To experimentally validate our algorithms, we built a simulation framework. We analyzed the impact of the direction and distance errors on the location estimation errors, and our experiments in diverse operational scenarios demonstrated that the average localization errors of our algorithms are near constant throughout the movement. We show how our algorithms can be utilized to create a stable and structured swarm of sensors without an underlying infrastructure or global positioning devices.

The remainder of this chapter is structured as follows. Section 4.1 discusses related work. Our localization algorithms are described in Section 4.2, while Section 4.3 outlines their use in coherent mobility scenarios. The main results of our experimentation are presented in Section 4.4, and Section 4.5 provides the concluding remarks.

## 4.1 Related work

Early research on sensor localization problems have primarily focused on static sensor networks [62]. Recently, however, more attention has been given to mobile environments. Problems in mobile sensor networks have been investigated mainly in conjunction with a particular positioning infrastructure (anchors, seed nodes, beacons) or under random

movement scenarios [20].

Low precision for close range and limited coverage (especially indoors) of GPS systems led to the investigation of GPS-free localization for mobile nodes. One common technique used is to exploit wireless communication. Bulusu *et al.* [17] use known reference points to send periodic beacon messages. By receiving beacons from these reference points, nodes can localize themselves. The accuracy of the localization depends on the distance to the reference points. Priyantha *et al.* [62] also use beacons for localization, but they assume the real locations of the reference points are unknown. The problem of calculating global geometry from local information is proved to be NP-hard [69]. For static nodes, and only using Euclidean distances, Bădoiu *et al.* [11] propose a constant factor, quasipolynomial-time approximation algorithm. The algorithm requires complete graph information, which results in substantial communication overheads [11] in mobile wireless networks.

In [20], *relative* localization in mobile sensor networks is accomplished through triangulation of neighbor nodes using a common one-hop neighbor. The authors propose algorithms for building a relative coordinate system based on a central node, or a dense group of nodes called Location Reference Group. Although this work is similar to ours in that it estimates positioning in a mobile environment without seed nodes, its primary focus is on negotiating a relative coordinate system for the entire network. While this solution finds applications in routing protocols, it is not applicable in mobility scenarios where directed motion is required, because the relative coordinate system used does not map to the real node positions.

In [43], a sequential Monte Carlo method is used to probabilistically estimate the locations of nodes in a network with a few seeds. Seeds are those nodes which know their precise location, through the use of GPS, for example. Due to the model's dependence on the prior estimates, the location errors are cumulative and a re-sampling step must be introduced. The re-sampling process requires each node to collect as much as fifty samples before a good estimate can be made. A method based on predictions is presented in [50],

where nodes in the network use a dead reckoning model to estimate the movements of all other nodes. Position information is adjusted for granularity so that distant pairs of nodes maintain less accurate position information than pairs which are closer to each other.

Concerning distance and motion detection error, Rayleigh fading may introduce significant errors due to the motion of the sensor in cases where signal strength is used for neighbor distance estimation. This problem is studied in [14], where the location estimation is based on power measurement of signals received from two anchored beacons with known locations. The authors explore how the speed of mobile nodes detrimentally affects their localization accuracy. The mechanisms introduced in [14] can complement our work to improve the neighbor distance measurement error for high speed sensors. Distance measurement methods are surveyed in [18]. The Time of Arrival (TOA) method finds the distance between a transmitter and a receiver through the use of one way propagation time. Time Difference of Arrival (TDOA) is another method to estimate the distance [34]. The TDOA method uses RF and ultrasound signals to estimate the distance accurately, at the expense of additional ultrasound transmitters and receivers.

## 4.2 Localization algorithms

In this section, we present our GPS-free localization algorithms. Our first algorithm, GDL, works under the following assumptions:

- Each node has a compass pointing North (or any other common reference direction).
- Nodes can measure the distance to their neighbors using a known range measurement method, such as Time of Arrival (TOA)[18], or Time Difference of Arrival (TDOA)[34].
- Motion actuators allow each node to move a specific distance in a specific direction (with respect to North).

- Actuator, compass and distance measurements are subject to errors caused by various real world disturbances such as wind, rough terrain, equipment failures etc.
- Other than the above, no additional positioning equipment or infrastructure is required.

We give details of our first algorithm in Section 4.2.1. In Section 4.2.2 we present our second algorithm, GCDL, which has the exact same assumptions listed above but does not require a compass.

### 4.2.1 GPS-Free Directed Localization (GDL)

The GDL algorithm makes use of a digital compass and achieves localization in a 3-stage process termed epoch. GDL consists of two sub-algorithms; Core localization and Verification. The core localization algorithm generates two possible relative positions for each neighbor that participates in the localization, and the verification algorithm uses a third neighbor to yield the correct final solution from these relative positions. Below, we give the details of these algorithms.

**Core localization algorithm:** The core localization algorithm works with variable length *epochs*, where each epoch involves three distinct stages:

1. Distance measurement between neighbors,
2. Individual movement of the nodes,
3. Exchange of direction and distance values for that epoch between neighbors.

Epochs are initiated by nodes whenever they need localization. Possible causes of localization are sudden increase or decrease in the number of neighbors, which hints clustering or partitioning in the network, and may require re-positioning of the nodes. We do not require any other continuity or pattern between epochs. We also do not assume anything about the temporal duration of the epochs. However, we assume that the nodes do not change their direction of movement within an epoch.



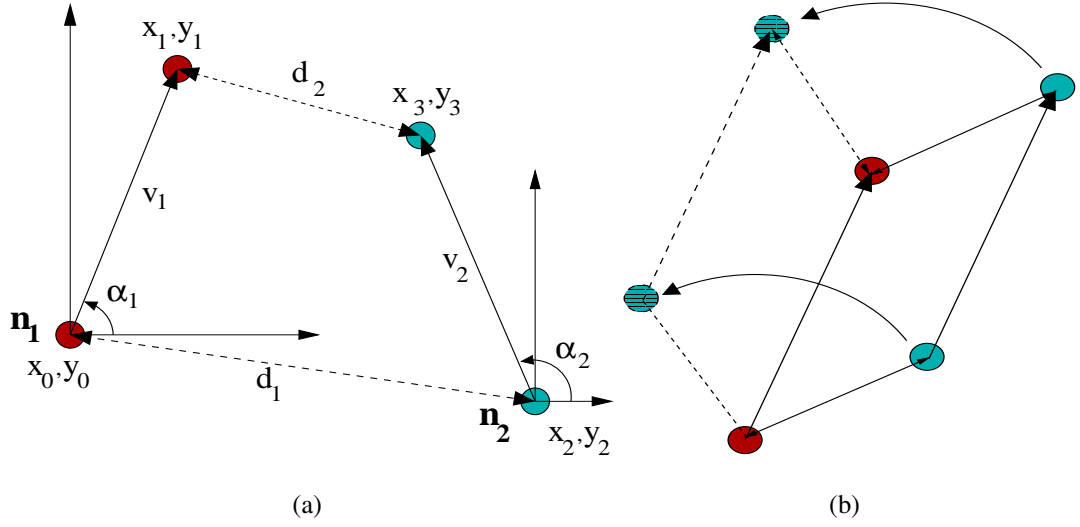


Figure 4.1: (a) Typical movement of two nodes, with angles and distances. (b) An example non-rigid geometry, where nodes move an equal distance in parallel. In the *equal parallel movement* exceptional configuration, localization is not possible because, geometrically, nodes can have infinite positions around each other.

A typical movement of two nodes  $n_1$  and  $n_2$  in an epoch is shown in Figure 4.1(a). At time  $t_1$ ,  $n_1$  is at position  $(x_0, y_0)$  and  $n_2$  at  $(x_2, y_2)$ , and the nodes measure the initial inter-distance  $d_1$ . Between time  $t_1$  and  $t_2$ , each node  $\{n_i \mid i = 1, 2\}$  moves in a direction  $\alpha_i$  and covers a distance  $v_i$ . At time  $t_2$ , the nodes, now at positions  $(x_1, y_1)$  and  $(x_3, y_3)$ , calculate their inter-distance  $d_2$  and exchange  $v_i$  and  $\alpha_i$  information. After receiving all the information, each node selects itself as the origin and calculates the position and direction of the other node, in its local coordinate system. To solve the equations in the local system of  $n_1$ , we choose the position  $(x_0, y_0)$  of  $n_1$  as the origin and write:

$$x_1 = v_1 \cos \alpha_1, \quad y_1 = v_1 \sin \alpha_1, \quad (4.1)$$

$$x_3 = x_2 + v_2 \cos \alpha_2, \quad y_3 = y_2 + v_2 \sin \alpha_2, \quad (4.2)$$

$$(x_3 - x_1)^2 + (y_3 - y_1)^2 = d_2^2, \quad x_2^2 + y_2^2 = d_1^2. \quad (4.3)$$

Substituting equations (4.1) and (4.2) into equation (4.3), we get:

$$x_2 A + y_2 B = C, \quad (4.4)$$

with the appropriate definitions:

$$A = v_2 \cos \alpha_2 - v_1 \cos \alpha_1, \quad B = v_2 \sin \alpha_2 - v_1 \sin \alpha_1,$$

$$C = \frac{1}{2} (d_2^2 - d_1^2 - v_1^2 - v_2^2 + 2v_1 v_2 \cos(\alpha_1 - \alpha_2)).$$

Substituting  $x_2 = (C - y_2 B)/A$  and  $y_2 = (C - x_2 A)/B$  into  $x_2^2 + y_2^2 = d_1^2$ , we get:

$$x_2^2 D - 2x_2 E + F = 0, \quad y_2^2 D - 2y_2 G + H = 0, \quad (4.5)$$

again with the appropriate definitions:

$$D = A^2 + B^2, \quad E = AC, \quad F = C^2 - d_1^2 B^2,$$

$$G = BC, \quad H = C^2 - d_1^2 A^2.$$

Note that the coefficient of  $x_2^2$  and  $y_2^2$  is the same in both equations (4.5), namely,  $D$ .

Using (4.5), each variable solves independently to

$$x_2 = \frac{E \pm \sqrt{E^2 - DF}}{D}, \quad y_2 = \frac{G \pm \sqrt{G^2 - DH}}{D} \quad (4.6)$$

and solutions can be paired up by using equation (4.4), as long as  $D \neq 0$ . In practice, one would compute either  $x_2$  or  $y_2$  using (4.5) and deduce the other variable using (4.4). When  $A = 0$  but  $B \neq 0$ , one would compute  $x_2$  using (4.6), and when  $A \neq 0$  but  $B = 0$ , one would compute  $y_2$  using (4.6) instead. If both  $A = B = 0$ , then  $D = 0$  and subsequently an *exceptional* configuration is formed; we discuss this case below.

The core localization algorithm to calculate the position of  $n_2$  from  $n_1$  is presented in Figure 4.2. Solving the equations, each node finds two possible positions for each of its neighbors. Since only one of these solutions is realistic (the other one is due to “symmetry”), each node has to complete a verification step, this time using an additional common neighbor ( $n_3$ ).

CORELOCALIZATION( $n_1, n_2, v_1, \alpha_1$ )

- 1:  $d_1 \leftarrow \text{inter-distance}(n_1, n_2)$
- 2: Move node  $n_1$  by  $v_1$  and  $\alpha_1$
- 3:  $d_2 \leftarrow \text{inter-distance}(n_1, n_2)$
- 4: Retrieve  $v_2$  and  $\alpha_2$  from  $n_2$
- 5: Calculate positions of  $n_2$  using equations (4.4),(4.5) and (4.6)

VERIFICATION(NEIGHBORLIST NL)

- 1: **for** each neighbor pair  $(m, n)$  in NL **do**
- 2:   **if**  $m$  and  $n$  are neighbors **then**
- 3:      $d_{m,n} \leftarrow \text{measured inter-distance}(m, n)$
- 4:     **for** each position pair  $\{m^i, n^j \mid i, j = 1, 2\}$  **do**
- 5:       Compute Euclidean distance  $D$  between  $m^i$  and  $n^j$
- 6:       **if**  $D = d_{m,n}$  **then**
- 7:          mark  $m^i$  and  $n^j$  as exact positions
- 8:       **end if**
- 9:     **end for**
- 10:  **end if**
- 11: **end for**

Figure 4.2: Core localization algorithm for  $n_1$ : calculates two possible positions for  $n_2$ . Verification algorithm evaluates the position estimations of neighbor nodes such that only 1 out of 4 position pairs validates the distance.

**Verification algorithm:** In Figure 4.2, we provide an algorithm that verifies a node's position using a third neighbor. This step is required to solve the ambiguity of two possible positions per neighbor calculated in core localization algorithm. After solving equations (4.4) and (4.6) in the previous section, node  $n_1$  has two position estimates  $\{n_j^{1,2} \mid j = 2, 3\}$  for each of its neighbors  $n_2$  and  $n_3$ . In order to find the positions and direction,  $n_1$  retrieves the distance between  $n_2$  and  $n_3$  ( $d_{2,3}$ ) from either one of these nodes, and simply finds the correct pair of positions  $\{n_j^{1,2} \mid j = 2, 3\}$  that has a matching distance.

For rigid geometries and configurations without errors, there can only be one pair verified. However, for configurations with errors, we relax the algorithm to select the pair with the closest distance value to  $d_{2,3}$ .

**Exceptional configurations:** The above localization algorithm works for rigid geometries where two possible positions per neighbor are estimated. Due to various real world disturbances and equipment errors, nodes do not always get a rigid geometry from their measurements. In this case equations (4.4) and (4.6) have no use. Any time the core algorithm cannot find meaningful results we reach what we term exceptional movement configurations. We distinguish two such configurations named *equal parallel movement* and *excessive error* configurations that we discuss below.

- *Equal parallel movement* configurations occur when  $D$  is equal to zero in equation (4.6). This also implies that  $A = 0$  and  $B = 0$  since  $D = A^2 + B^2$ . An example of *equal parallel movement* configuration is shown in Figure 4.1(b). In this case, the nodes move in parallel and keep the exact same distances ( $d$  and  $v$ ) between them, so that node  $n_2$  can be anywhere on a circle at a distance  $d$  away from node  $n_1$ , and vice versa. The geometry is not rigid and infinitely many possible solutions exist for both neighbors.
- The second exceptional configuration is that of *excessive error*. The main sources of error in our algorithm occur due to distance, actuator and compass measurement

inaccuracies. When highly erroneous  $d$ ,  $v$  and  $\alpha$  values create a non-rigid geometry, such that  $E^2 - DF < 0$  or  $G^2 - DH < 0$  in equation (4.6), our core algorithm cannot localize  $n_1$  and  $n_2$ .

Although we cannot entirely avoid the above exceptional configurations, the core localization part of GDL algorithm can readily detect them. Once detection takes place, nodes can skip that epoch and can make necessary adjustments (e.g. random changes) to their speed and direction to avoid the same ill-configuration in the next epoch.

### 4.2.2 GPS and Compass-Free Directed Localization (GCDL)

Additional cost on hardware and possible physical conditions altering magnetic field restrict the use of compass in certain hostile environments such as disaster areas. To enhance versatility, we relax the requirement for a compass and achieve localization using an algorithm that controls the mobility of the nodes. The main idea is to divide the nodes into two groups, *blue* (dark) and *red* (light), and move only one of these groups while the other group is stationary. We show that, through the use of geometric properties, we can localize the neighbors in a 2-step motion for each group. After localization, nodes can agree on a common north, which essentially has the same effect as having a compass. Common north resolution also enables the nodes to perform coherent movement as a network, which is one of our main goals. Furthermore, by relaxing the requirement for a compass, we relieve our GCDL algorithm from compass related failures such as measurement or equipment errors, and improve the robustness of our approach.

We outline our 2-step motion algorithm (Figure 4.3) using a *blue* (dark) and a *red* (light) node. The *blue* node is stationary, and the *red* node performs a 2-step motion to localize the exact position of the *blue* node. Each time the *red* node communicates with the *blue* node, a virtual communication circle is formed. In order to exactly find the position of the *blue* node, three circles are needed. Therefore, by keeping track of its own movement distance, in the worst case, the *red* node gathers enough data to localize the *blue* node only

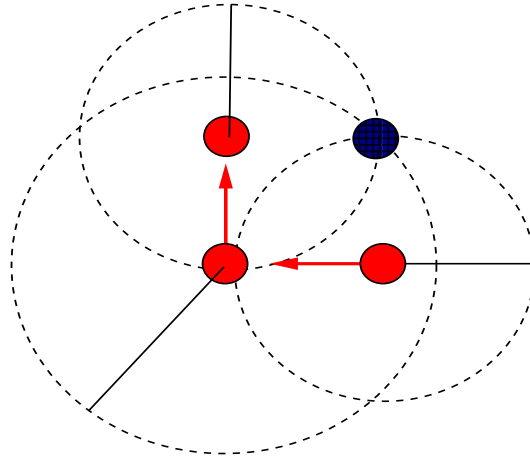


Figure 4.3: 2-step motion for localizing the *blue* node.

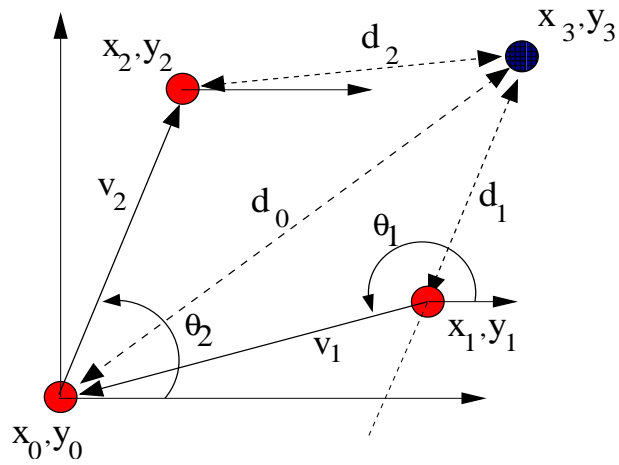


Figure 4.4: Geometry of the GCDL localization showing the *blue* (dark) node stationary at position  $x_3, y_3$ , and *red* (light) node performing 2-step motion from  $x_1, y_1$  to positions  $x_0, y_0$  and  $x_2, y_2$  in order to localize the *blue* node.

after 2-step motion. Note that if the movement of the *red* node is directly towards (or away from) the *blue* node, one step of the motion is enough for localization, since in this case the geometry forms two mutually tangent circles within each other, which intersect at the exact location of the *blue* node.

In order to outline the geometric calculations necessary for localization after 2-step motion we assume the *blue* node in Figure 4.4 is stationary at position  $(x_3, y_3)$ , and the *red* node moves to positions  $(x_1, y_1)$ ,  $(x_0, y_0)$ , and  $(x_2, y_2)$  respectively. We also assume that each node has a local coordinate system, based on an arbitrary north, and nodes can mechanically track their movements relative to this local coordinate system. For example nodes can mechanically turn  $90^\circ$  based on their local north without requiring a compass. The movement of the *red* node is represented by a  $\langle \text{distance}, \text{angle} \rangle$  pair. Here, the angle is relative to the local coordinate system of the *blue* node. The first movement from position  $(x_1, y_1)$  to  $(x_0, y_0)$  is represented as  $\langle v_1, \theta_1 \rangle$ , and the second movement from position  $(x_0, y_0)$  to  $(x_2, y_2)$  is represented by  $\langle v_2, \theta_2 \rangle$  pair. For each position of the *red* node, we write the following formulas:

$$(x_3 - x_1)^2 + (y_3 - y_1)^2 = d_1^2, \quad (4.7)$$

$$(x_3 - x_2)^2 + (y_3 - y_2)^2 = d_2^2, \quad (4.8)$$

$$x_3^2 + y_3^2 = d_0^2. \quad (4.9)$$

We write the first  $(x_1, y_1)$  and last positions  $(x_2, y_2)$  of the *red* node as:

$$x_1 = v_1 \cos(\theta_1 - \pi), \quad (4.10)$$

$$y_1 = v_1 \sin(\theta_1 - \pi), \quad (4.11)$$

$$x_2 = v_2 \cos \theta_2, \quad (4.12)$$

$$y_2 = v_2 \sin \theta_2. \quad (4.13)$$

Rewriting Eq.(4.7) and substituting Eqs.(4.9,4.10, 4.11) we get:

$$(x_3 - x_1)^2 + (y_3 - y_1)^2 = d_1^2,$$

$$x_3^2 - 2x_1x_3 + x_1^2 + y_3^2 - 2y_2y_3 + y_1^2 = d_1^2,$$

Observing that  $x_1^2 + y_1^2 = v_1^2$ ,

$$d_0^2 - y_3^2 - 2v_1x_3 \cos(\theta_1 - \pi) + v_1^2 + y_3^2 - 2v_1y_3 \sin(\theta_1 - \pi) = d_1^2,$$

We calculate  $x_3$  as:

$$x_3 = \frac{d_0^2 + v_1^2 - 2v_1y_3 \sin(\theta_1 - \pi) - d_1^2}{2v_1 \cos(\theta_1 - \pi)}, \quad (4.14)$$

Using  $x_3$  in Eq.(4.8), and substituting Eqs.(4.12,4.13) we get:

$$(x_3 - x_2)^2 + (y_3 - y_2)^2 = d_2^2,$$

$$x_3^2 - 2x_2x_3 + x_2^2 + y_3^2 - 2y_2y_3 + y_2^2 = d_2^2,$$

Again observing that  $x_2^2 + y_2^2 = v_2^2$ ,

$$d_0^2 - \frac{d_0^2 + v_1^2 - d_1^2 - 2v_1y_3 \sin(\theta_1 - \pi)}{v_1 \cos(\theta_1 - \pi)} v_2 \cos \theta_2 + v_2^2 - 2v_2y_3 \sin \theta_2 = d_2^2, \quad (4.15)$$

We can now calculate  $y_3$  from Eq.(4.15). To ease the presentation in Eq.(4.15), we define  $B$  as:

$$B = \frac{v_2 \cos \theta_2}{\cos(\theta_1 - \pi)} 2 \sin(\theta_1 - \pi) - 2v_2 \sin \theta_2,$$

if we simplify,  $B$  becomes:

$$B = \frac{2v_2 \sin(\theta_1 - (\pi + \theta_2))}{\cos(\theta_1 - \pi)}, \quad (4.16)$$

Again, to ease the presentation in Eq.(4.15), we define  $A$  as:

$$A = d_2^2 - d_0^2 - v_2^2 + \frac{v_2 \cos \theta_2}{v_1 \cos(\theta_1 - \pi)} (d_0^2 + v_1^2 - d_1^2), \quad (4.17)$$

where using Eqs.(4.16, and 4.17)  $y_3$  becomes:

$$y_3 = \frac{A}{B} \quad (4.18)$$

Solving equations (4.14) and (4.18), determines how the *red* node localizes the *blue* node. Next, the *red* and the *blue* nodes switch roles, and the *blue* node moves while the *red* node stays stationary. After both groups of nodes complete their 2-step motion, they agree on a common north for the entire network, which we describe in Section 4.2.2.



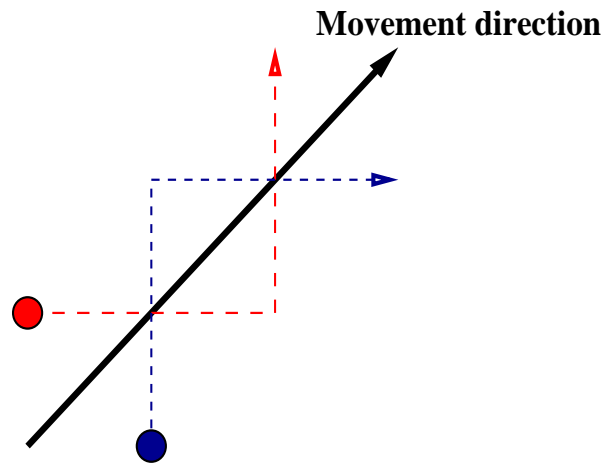


Figure 4.5: Zig-zag motion of nodes towards movement direction.

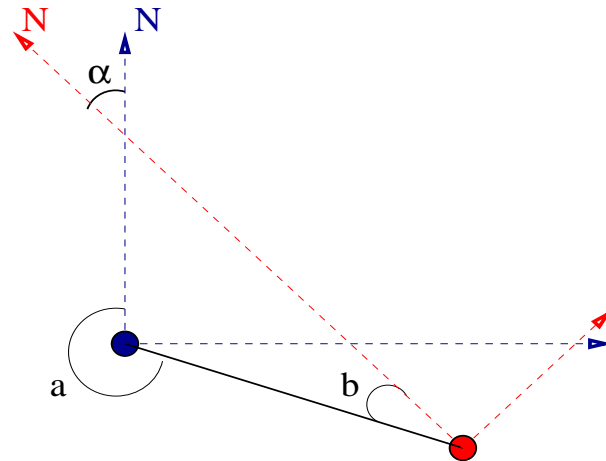


Figure 4.6: Detecting and correcting the skew between local coordinate systems of neighbors. When the relative positions of the nodes to each other are  $a$  and  $b$ , the skew in their local coordinate systems is  $\alpha$ .

### Lock-step movement

For applications where the mobile sensor nodes are required to move as a cohort from one region to another, we configure our GCDL algorithm for pseudo-directional movement. We randomly assign *red* and *blue* colors to nodes at each iteration to uniformly color the nodes in the swarm. Once the direction of movement is known by all nodes, each group does a *zig-zag* movement following the direction, as shown in Figure 4.5. A *zig-zag* movement fits with our algorithms requirement of the 2-step motion, while still allowing the node to move towards a certain direction. We would like to highlight here that the *zig-zag* movement can be performed with different step movement angles. In this chapter, we use orthogonal movements as those shown in Figure 4.5 to simplify the presentation.

### Selecting a common north

In GCDL, nodes coordinate their movement based on a pseudo-north, since no compass is used. As the pseudo-north is selected arbitrarily by each node, and altered by the environmental conditions throughout the movement, nodes have to agree on a common north with their neighbors to move as a cohort. Once both group of nodes complete their lock-step motion, neighbor nodes can further communicate to agree on a common north for all. In order to achieve this goal, each node exchanges calculated position information of each of its one-hop neighbors, based on its local coordinate system. Each node, by cross examining its neighbor's position relative to itself ( $\angle a$  in Figure 4.6) and its own position relative to that specific neighbor ( $\angle b$ ), calculates the skew in local coordinate systems ( $\angle \alpha$  in Figure 4.6). Once the skew between local coordinate systems is detected, the nodes can use two methods in order to agree on a common coordinate system: (1) use a proactive approach and force the entire swarm to agree on a single coordinate system by using a hierarchy (e.g. TAG tree [54]). (2) use a reactive method and store only the variance ( $\alpha$ ) for each neighbor, and initiate a local correction each time a direction information is received from neighbors. The use of the proactive or the reactive approaches are specific

to the characteristic of the network and the application. Both approaches can be used with our GCDL algorithm.

In this section, we presented our localization algorithms. GDL performs localization in single step of the motion by use of a compass on each node, while GCDL performs localization without the use of a compass, following a 2-step motion.

### 4.3 An example sensor network mobility algorithm

Our algorithms are most useful in mobile applications where the entire network must move in a specific path in order to accomplish a goal. To analyze the behavior of our localization algorithms in a realistic mobility scenario, we adapt a mobility model based on the Reference Point Group Mobility (RPGM) model [42]. Although we considered a range of mobility models[19], we decided to base our analysis on RPGM due to its generality and simplicity. Here, the random motion of the individual nodes is modeled in relation to a randomly chosen directional motion of the entire group. Each node in the group moves randomly around a fixed reference point and the entire group of reference points moves along the group's logical center. Our localization algorithms computes locations and orientations for nodes and their neighbors. In that respect, we further generalize the RPGM model so as to make individual sensors independent of the reference points. Furthermore, because our sensor network can maintain a semi-rigid structure based solely on local positioning, it is unnecessary for nodes to be aware of the group's center and only the destination point must be specified. It is possible to remove the reference points because the individual random motion within the group is contextualized by the random motion of a node's immediate (one-hop) neighbors. In that sense, the neighbors represent the reference points of motion.

The mobility algorithm we use for directed motion is presented in Figure 4.7. The network moves with respect to a direction vector  $\vec{D}$ . To maintain a semi-rigid formation without disconnecting the network, we impose a minimum neighbor count  $k$  that each node strives to attain. This is a best-effort algorithm where nodes attempt to maintain

```

MOVE_NODE(NODE N, NEIGHBORLIST NL,
DIRECTIONVECTOR  $\vec{D}$ , INT  $k$ , RANGEFACTOR RF)

1:  $\vec{V} \leftarrow 0$ 
2:  $count \leftarrow 0$ 
3: for each localized neighbor  $n$  in NL do
4:   /*  $\vec{u}_{N,n}$  is the vector from  $N$  to  $n$  */
5:    $\vec{V} \leftarrow \vec{V} + \vec{u}_{N,n}$ 
6:    $count \leftarrow count + 1$ 
7: end for
8: if  $count < k$  then
9:    $RF \leftarrow RF / 2$ 
10: end if
11:  $\vec{V} \leftarrow (RF * range(N) * \vec{V} + \vec{D}) / (count + 1)$ 
12: Move node  $N$  by  $\vec{V}$ 

```

Figure 4.7: The mobility algorithm we use for directed motion.  $RF$  is the fraction of the wireless range; used by nodes as an ideal distance with their neighbors, and  $range(NodeN)$  is the wireless range of the given node.

a neighbor distance that is a fraction of their wireless range. The neighbor distance is adjusted dynamically with the number of neighbors so that nodes with neighbors fewer than  $k$  stay closer while still moving with the network. This avoids network partitioning especially at the perimeter of the network. Localization of the one-hop neighbors is a prerequisite for the mobility algorithm to run efficiently. However, the algorithm does not strictly require all neighbors to be localized, it considers only the localized neighbors and performs the necessary calculations based on these. The  $range(Node N)$  function returns the wireless range in distance of the given node. When a boundary is reached, a new direction of movement must be chosen. In our simulations, we implement this as a ricochet off the boundary surface.

The benefit of our approach is that while an initial direction of motion is specified for the group, the structure of the network remains cohesive but independent. An example application of this approach is a swarm of mobile sensors which move in a general pattern with a specific goal. For example, a swarm of mobile sensors may move in a zig-zag pattern, with the goal to discover an oil spill and cover the contaminated area once it is found. In this example, only a virtual boundary must be specified and the network of sensors will maintain sufficient proximity to communicate, while covering the area.

The mobility algorithm presented in this section is a general network movement algorithm that requires only local position information. Our localization algorithms require each node to communicate only with its direct neighbors thus no message broadcast is required. In this respect, other mobility algorithms can be plugged in to perform various tasks using our localization algorithms.

## **4.4 Experiments**

In this section, we evaluate our algorithms in three different type of experimental settings. First, in Section 4.4.1 we evaluate properties specific to our GDL algorithm under various simulated error settings, and show that the environment errors do not dominate the performance of our algorithm. Later, in Section 4.4.2 we evaluate our GCDL algorithm under various errors and observe how the errors affect the accuracy of the localization and common north resolution algorithms. Finally, in Section 4.4.3 we compare both our algorithms against an Absolute Positioning algorithm in various random and directed mobility scenarios, and observe the effects of cumulative errors on the localization algorithms and the coherent movement of the swarm.

### **4.4.1 Evaluation of the GDL algorithm**

We initially present results from the GDL algorithm under ideal conditions, without measurement errors. Subsequently, we introduce independent errors on angle and dis-

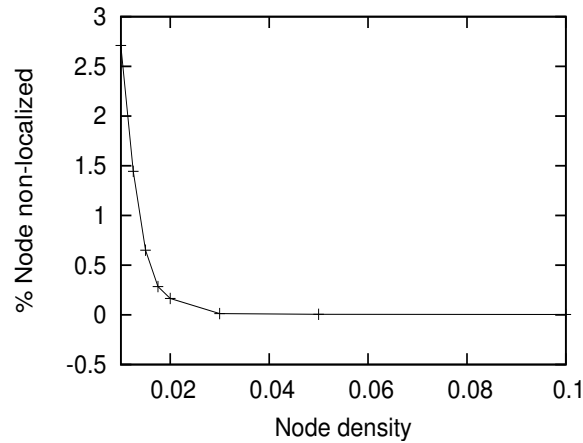
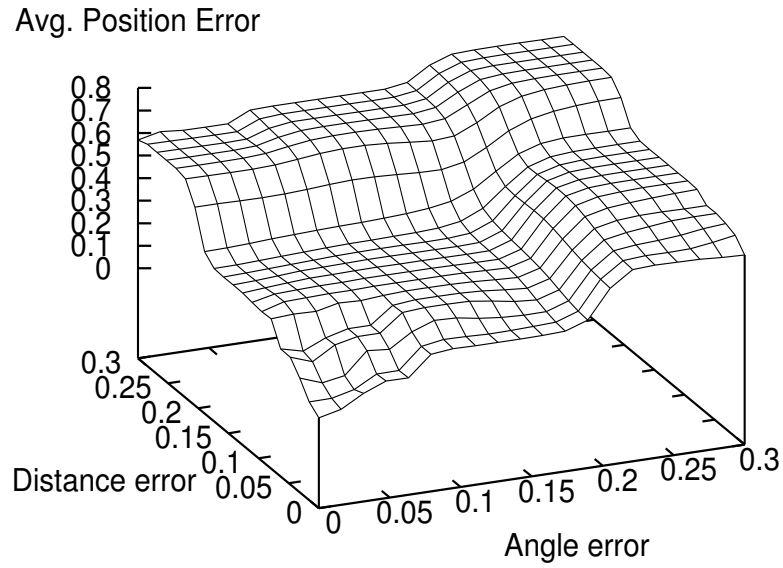


Figure 4.8: Percent of non-localized nodes for different node densities.

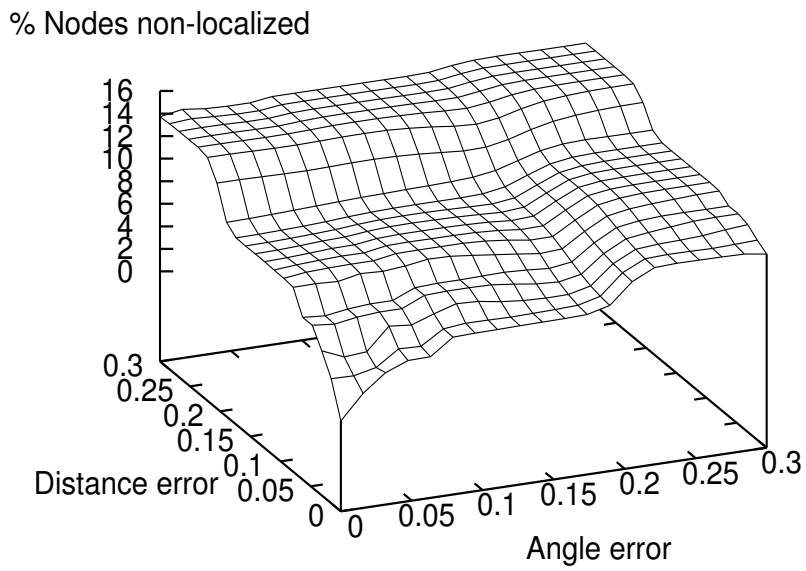
tance measurements to simulate real world disturbances.

**Experiments under ideal conditions** In this experiment we simulate nodes randomly placed in a 100x100 area under a uniform spacial distribution. Each simulation is run for 100 epochs, and the results are averaged. At each epoch, nodes perform a random walk with random speed  $[0, 5)$ , random angle  $[0, 2\pi)$  and fixed radio range of 6. Node density represents the number of nodes over the total deployment area. GDL requires two neighbors to accurately find neighbor positions. Figure 4.8 displays the percentage of nodes, whose positions are not calculated accurately for different node densities. For small node densities, we observe that not all nodes can be localized. The reason is that nodes do not have neighbors to calculate positions, or do not have common neighbors. As we can see from Figure 4.8, the percent of non-localized nodes approaches zero for densities greater than 0.02. From this graph we can conclude that our algorithm calculates node positions for dense networks, and introduces minor node localization failures, less than as 3%, for sparse networks.

**Introducing measurement errors** We now relax the ideal condition assumption and introduce errors on distance and angle measurements. In the real world, measurements may



(a)



(b)

Figure 4.9: Effects of angle and distance measurement noise on position error (a), and percent of non-localized nodes (b), for GDL

be quite inaccurate due to weather, terrain conditions and equipment failures. To simulate these errors, we add uniform random noise to all our measurements. For distance measures we add percent error relative to the measured value, and for angle measures we add absolute percent error (percent of  $2\pi$ ) to the measured value. Introducing the errors changes our algorithm's behavior in one of two ways: (1) the algorithm calculates the positions with limited accuracy; or (2) *excessive error* configurations (defined in Section 4.2) prevent the algorithm from localizing some of the nodes. Figure 4.9 (a) shows the average position error of our algorithm for different values of noise on angle and distance measurements. The effects of *excessive error* configurations on our algorithm appear in Figure 4.9 (b). From this figure we can see that even with 30% noise on angle and distance measures, which is a quite high error rate for real world conditions, the number of non-localized nodes is at most 16%. These results indicate that our GDL algorithm provides sufficient node localization. We further corroborate this claim by carrying out tests in random and directed movement scenarios in Section 4.4.3.

In order to evaluate the effects of movement speed and wireless range we tested our GDL algorithm under high noise, with a fixed wireless range (10) and variable speed values for nodes.<sup>1</sup> We apply an upper limit on the speed, such that the nodes do not move a distance greater than their wireless range per epoch. In any sensor network scenario, if a node moves by a distance greater than its wireless range in one epoch, it is highly probable that its neighborhood will change at each step, which would make it impossible to localize. We can see in Figure 4.10 that the localization error of our algorithm is nearly constant for increasing speed of nodes. The maximum speed supported by our algorithm is the wireless range distance per epoch which is 10 units per epoch in this experiments.

---

<sup>1</sup>As our GCDL algorithm performs a lock-step motion, we exclude GCDL from this movement speed experiment.



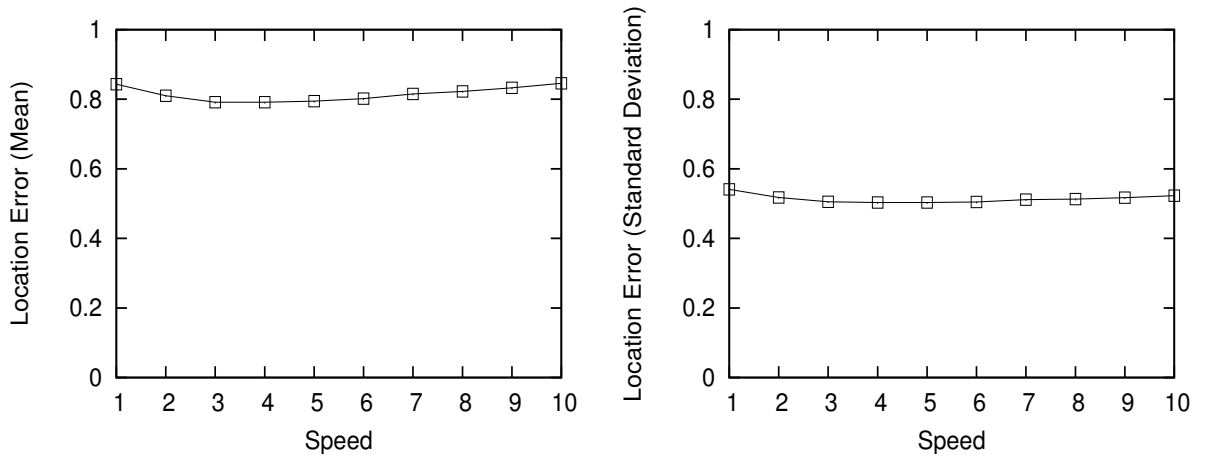


Figure 4.10: Mean and standard deviation of position error of GDL vs. speed of nodes with a wireless range of 10 units and speed measured in units per epoch.

#### 4.4.2 Evaluation of the GCDL algorithm

In this section, we evaluate the affects of various environment errors on the common north resolution method of our GCDL algorithm. We use the average difference between the real and the calculated norths of each neighbor per epoch as the metric to measure the common north error of the swarm.

We first evaluate whether the duration of the movement affects the common north error. We simulate 100 nodes in a 100x100 area. In random motion, nodes can cover at most 5 units and have a fixed radio range of 15. We assume that in directed motion nodes move at a maximum speed of 3 units per epoch and the radio range is set to 5. All experiments are performed 100 times, and the average values are displayed. We test our algorithm for two different uniform random noise levels: high and low as well as in random motion and directed motion scenarios. High noise level is up to  $\pm 30\%$  of distance measurements and up to  $\pm 2\pi/10$  of angle measurements. Low noise level is up to  $\pm 3\%$  of distance measurements and up to  $\pm 2\pi/100$  of angle measurements. In this experiment, after each epoch, nodes resolve the common north variance with their neighbors and the average error per node is calculated. Figure 4.11 shows that for both random and directed movement scenarios, the

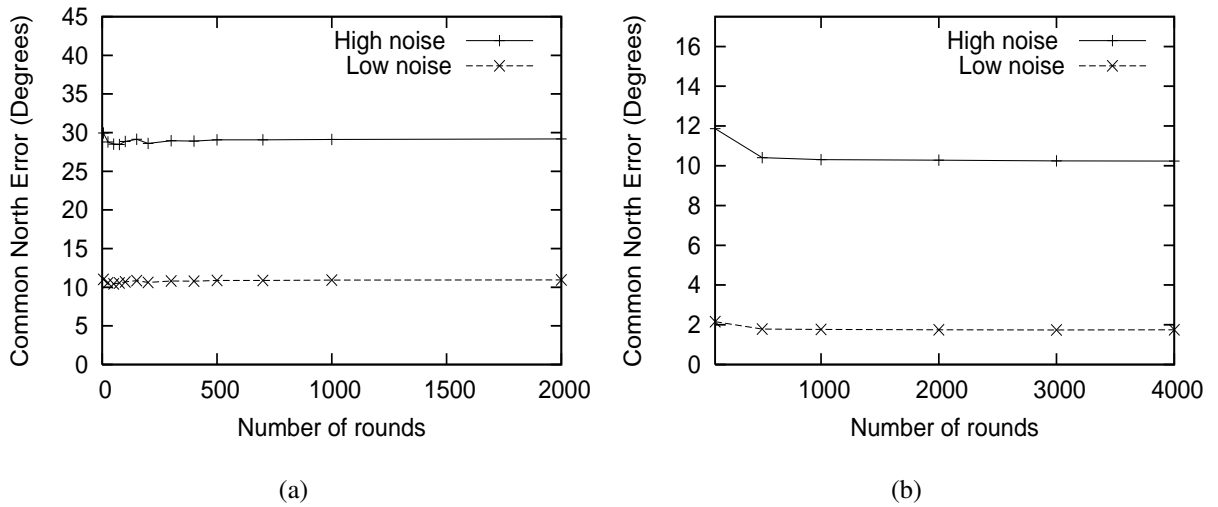
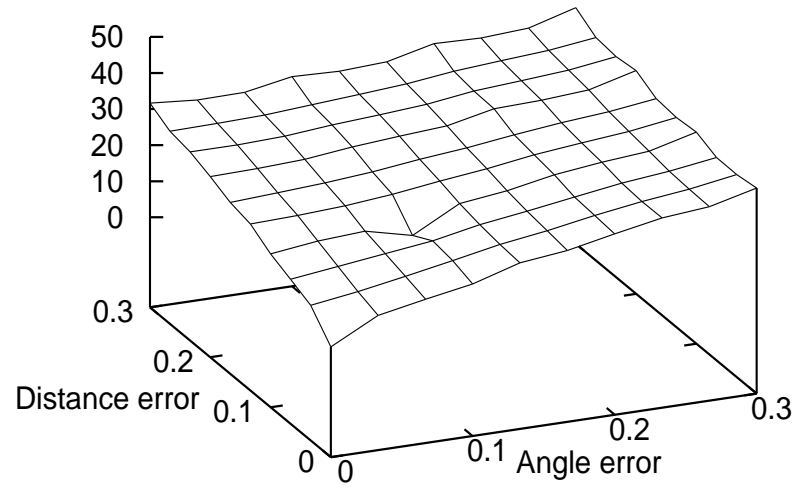


Figure 4.11: Effect of number of epochs on selected common north angle, (a) random motion, and (b) directed motion, for GCDL

average error on common north remains constant throughout the total number of epochs. This is expected as we do not store any information other than the current motion. All calculations are performed from scratch based on the new neighborhood after the 2-step motion. Thus, GCDL is free from incremental errors.

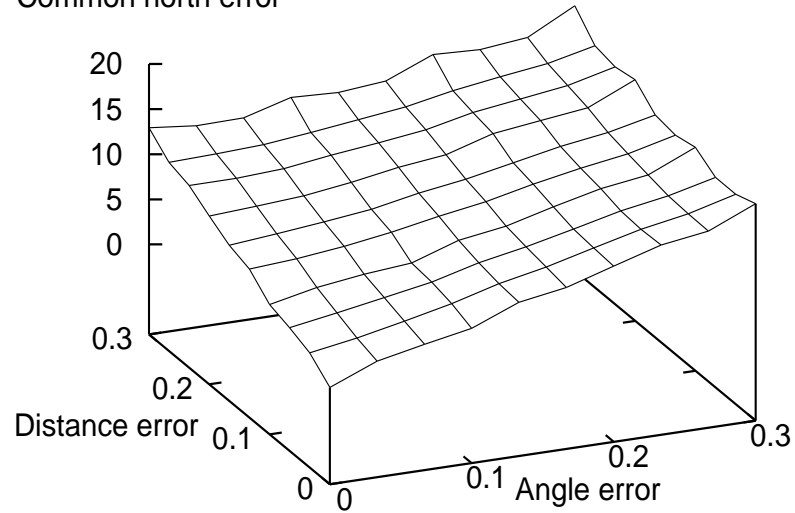
Having established that the common north error is not affected by the number of epochs, we vary the distance and angle measurement noise and see how these changes affect the common north error in Figure 4.12. In this figure, we can observe that the amount of noise on the angle and the distance measurements both affect the common north error of the GCDL algorithm. The common north error is zero for environments free of error, and the error increases linearly with the angle and distance errors when they are introduced. For a low noise level, the common north errors are approximately  $10^\circ$  and  $2^\circ$  respectively for random and directed motion. For a high noise level, the errors become approximately  $34^\circ$  and  $14^\circ$ , respectively.

Common north error



(a)

Common north error



(b)

Figure 4.12: Effects of angle and distance measurement noise on selected common north error in degrees, (a) random motion, and (b) directed motion, for GCDL. The distance and angle noise axes show the percent error over the actual measurement.

### 4.4.3 Comparison with an absolute positioning algorithm

In previous sections, we present experiments to test the specific properties of each of our algorithms. In this section, we compare our algorithms with an Absolute Positioning algorithm. In such an algorithm, we assume that nodes know their initial positions in the deployment area, thanks to an anchor point or any other global positioning infrastructure. We also assume that once nodes get their initial position, they do not receive any additional positioning information, relative or absolute. To this effect, nodes keep track of their own movements. By exchanging location information with immediate neighbors, each node is able to keep track of the positions of others. This scenario occurs when nodes are deployed from a known position and asked to explore a possibly remote area where they cannot maintain communication with the anchor at the deployment position.

We simulate two different mobility scenarios. The first is based on random movement, where 100 nodes with fixed radio range 15 can cover a distance of at most 5 units per epoch. The second scenario is the directed movement described in Section 4.3. Nodes sweep the area in a zig-zag manner, with radio range 5 and maximum per-epoch distance 3. An example trajectory of the nodes in the directed movement scenario is shown in Figure 4.16. There is no global path information available, rather, the nodes detect the boundaries of the environment and make movement decisions as a response to these environmental readings.

In Figures 4.13 and 4.14, we show the average errors for all algorithms over the progression of up to 4,000 rounds, for two different uniform random noise levels: high and low, in random motion (Figure 4.13) and directed motion (Figure 4.14) scenarios. Since our algorithms calculate distances within each epoch and do not use any cumulative data, the error of our algorithms are nearly constant over the number of epochs, for both scenarios. Although the absolute positioning algorithm starts with a low error value, small measurement errors accumulate over each epoch and cause an ever-increasing error. The mean error of the GDL is as much as 2 times and the mean error of the GCDL is as much as 10 times lower than the mean error of the absolute positioning algorithm. The high

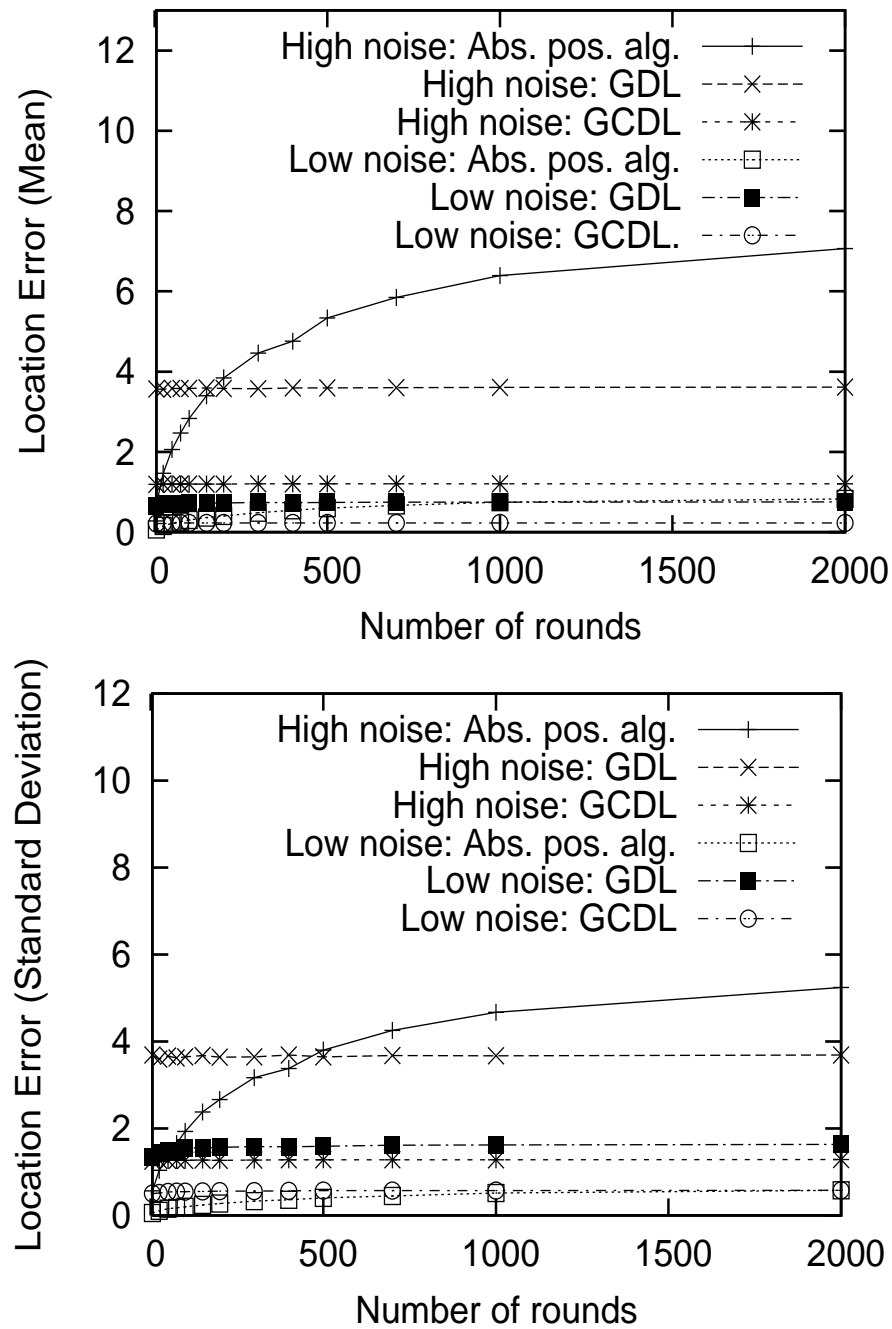


Figure 4.13: Mean and standard deviation of position error vs. number of epochs for our algorithms and the absolute positioning algorithm performing random movement, using different levels of noise. The error of the absolute positioning algorithm increases with the number of epochs while the error of our algorithms are almost constant, which is attributed to the memoryless property of our algorithms.

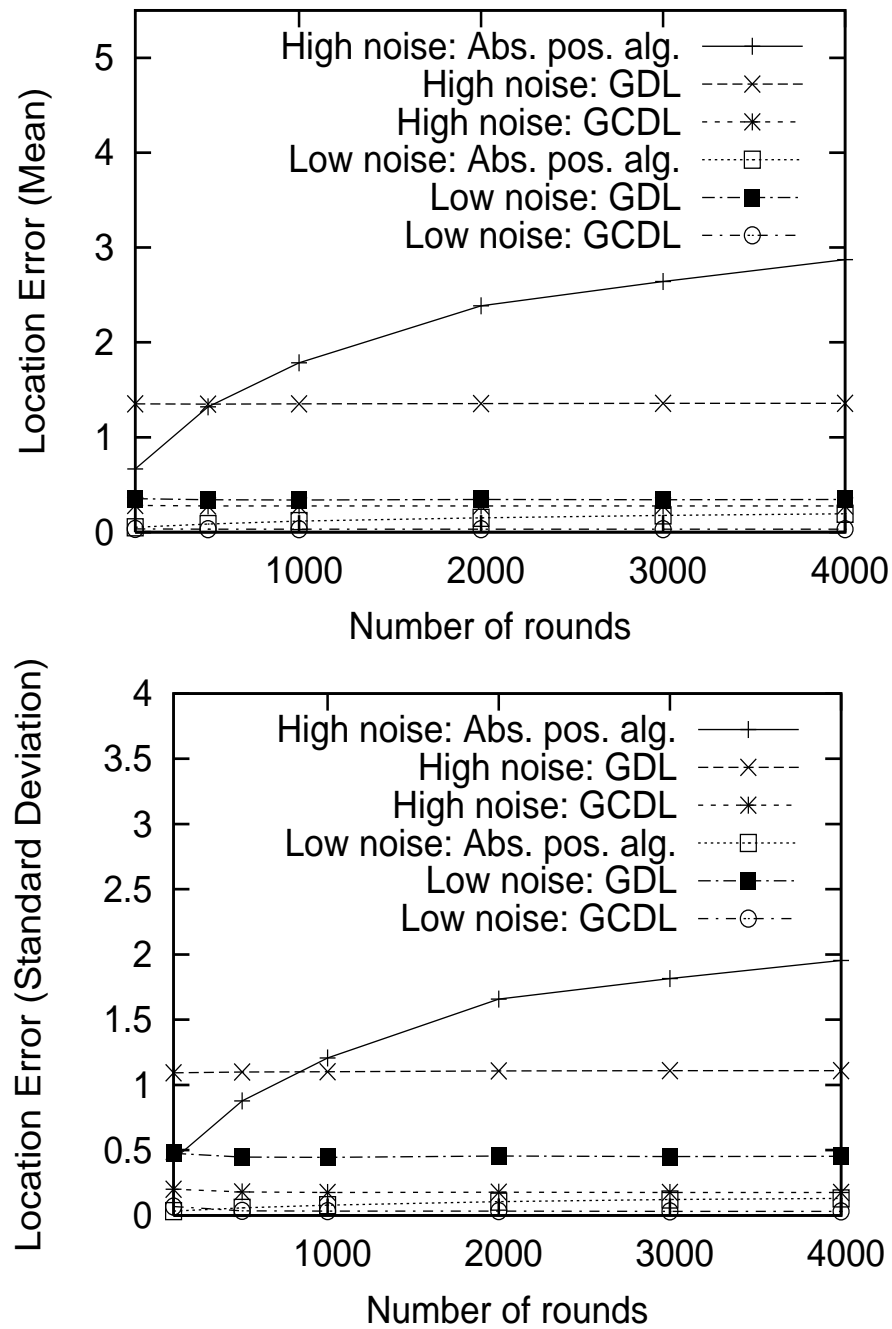


Figure 4.14: Mean and standard deviation of position error vs. number of epochs for our algorithms and the absolute positioning algorithm performing directed movement, using different levels of noise. The error of the absolute positioning algorithm increases with the number of epochs while the error of our algorithms is almost constant, which is attributed to the memoryless property of our algorithms.

Movement	GDL	GCDL
Random	1.95 ms	1.08 ms
Directed	2.06 ms	1.12 ms

Figure 4.15: Average time in milliseconds (ms) spent for localization calculations in core GDL and GCDL algorithms.

error values in the absolute positioning algorithm reflect on the effects of cumulative errors and show that it is not a robust solution for high noise scenarios. On the other hand, our localization algorithms provide consistent behavior in the above-mentioned scenarios.

Figure 4.15 presents the average CPU time each node spent calculating Equations (4.4) and (4.6) in GDL and Equations (4.14) and (4.18) in GCDL, for both random and directed mobility scenarios. Since we do not have access to the real sensor hardware, we ran the experiments on a Pentium IV 3 GHz machine and present the average time spent per localization in order to compare the performance of our algorithms. As we can see from the figure, the CPU overhead is less for GCDL compared to GDL, which is a direct result of the number of calculations required by each algorithm.

In Figures 4.17, 4.18 and 4.19 we present the snapshots of the simulations of the absolute positioning algorithm, our GDL and GCDL algorithms, respectively, performing a zig-zag directed movement (as in Figure 4.16) under high noise. All simulations use the same movement algorithm as described in Section 4.3. Because of the cumulative errors, the absolute positioning algorithm is not capable of maintaining the topology of the network and becomes disconnected. On the other hand, our algorithms maintain connectivity at all times while forming a nice semi-rigid topology, which complies with one of our main goals to enable coherent movement of nodes as a swarm even under high noise. The snapshot results for GCDL algorithm is more "packed" than its GDL counterpart. This behavior is caused by the separation of nodes into two groups. Since only nodes in opposite groups localize each other, each node maintains its distance to the nodes in the opposite group and

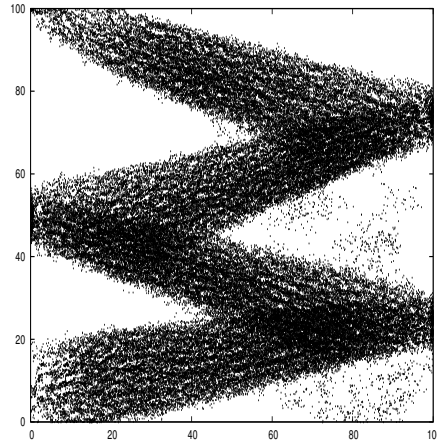


Figure 4.16: Directed trajectory of nodes performing zig-zag movement.

tends to stay closer with nodes in its group. Even in this case, GCDL succeeds to maintain the connectivity and semi-rigid topology of the swarm. The results in Figures 4.13, 4.14, 4.18, and 4.19 also support our claim that even under high noise settings, environmental errors do not deteriorate our algorithm's behavior; effects of these errors are constant through epochs. As seen in Figure 4.18, occasionally a few nodes (two in this case) disconnect from the network. This happens when nodes near the network's perimeter cannot be localized. Although the number of these nodes is small, they can be further controlled by forcing stricter  $k$ -neighborhood rules in the mobility algorithm.

## 4.5 Concluding remarks

We propose two algorithms to address the directional node localization problem in wireless sensor networks. Our directional localization algorithms enable nodes to coordinate their movement relative to their one-hop neighbors, and maintain a semi-rigid structure throughout the movement of the swarm. During mobility, errors from measurement devices used by the nodes or real world disturbances tend to accumulate overtime and affect the structure of the swarm, eventually disorganizing it. To avoid this phenomenon, our algorithms perform localization in a few epochs of the node movement and work only



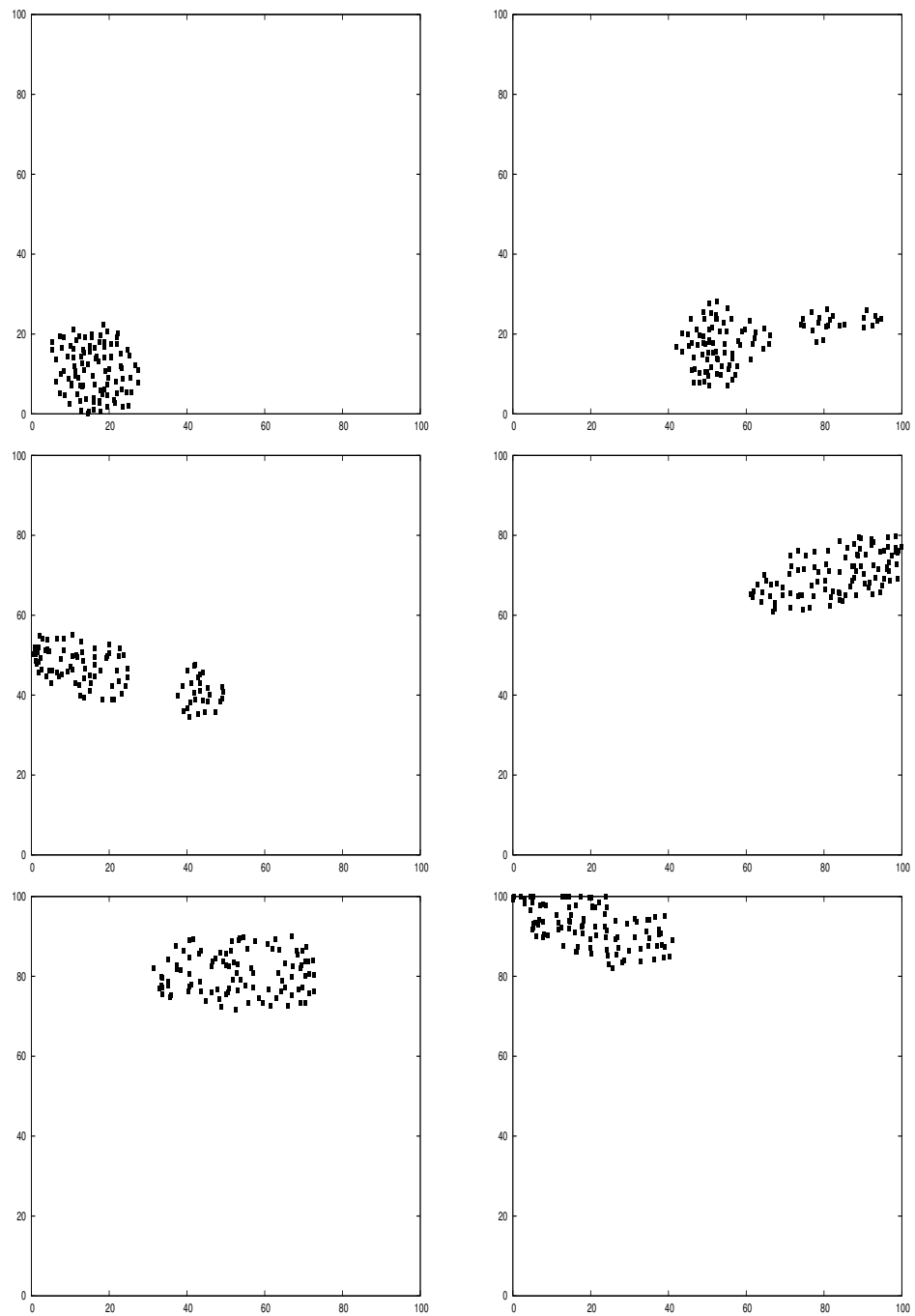


Figure 4.17: Snapshots of absolute positioning algorithm performing directed motion in Figure 4.16.

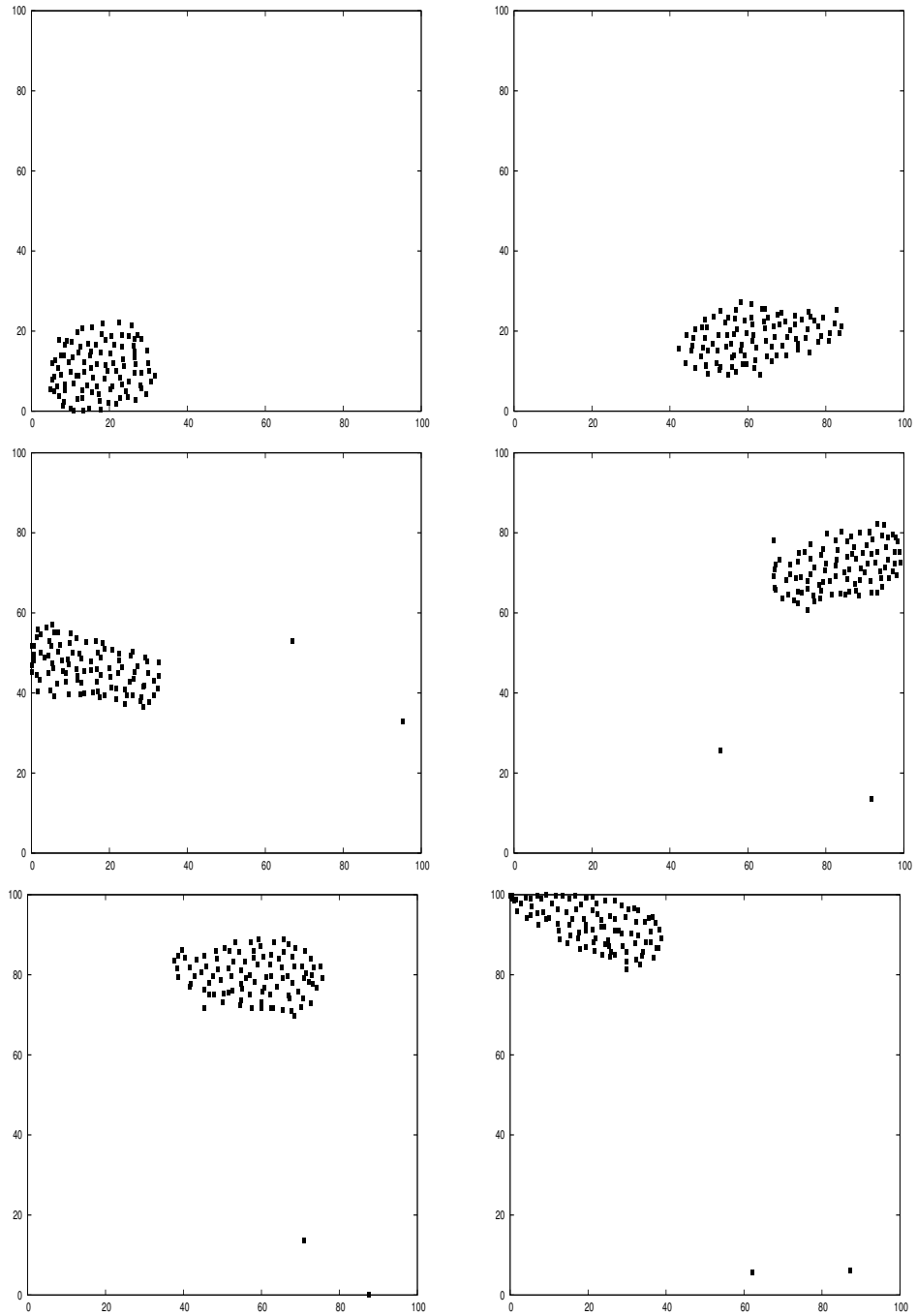


Figure 4.18: Snapshots of GDL algorithm performing directed motion in Figure 4.16.

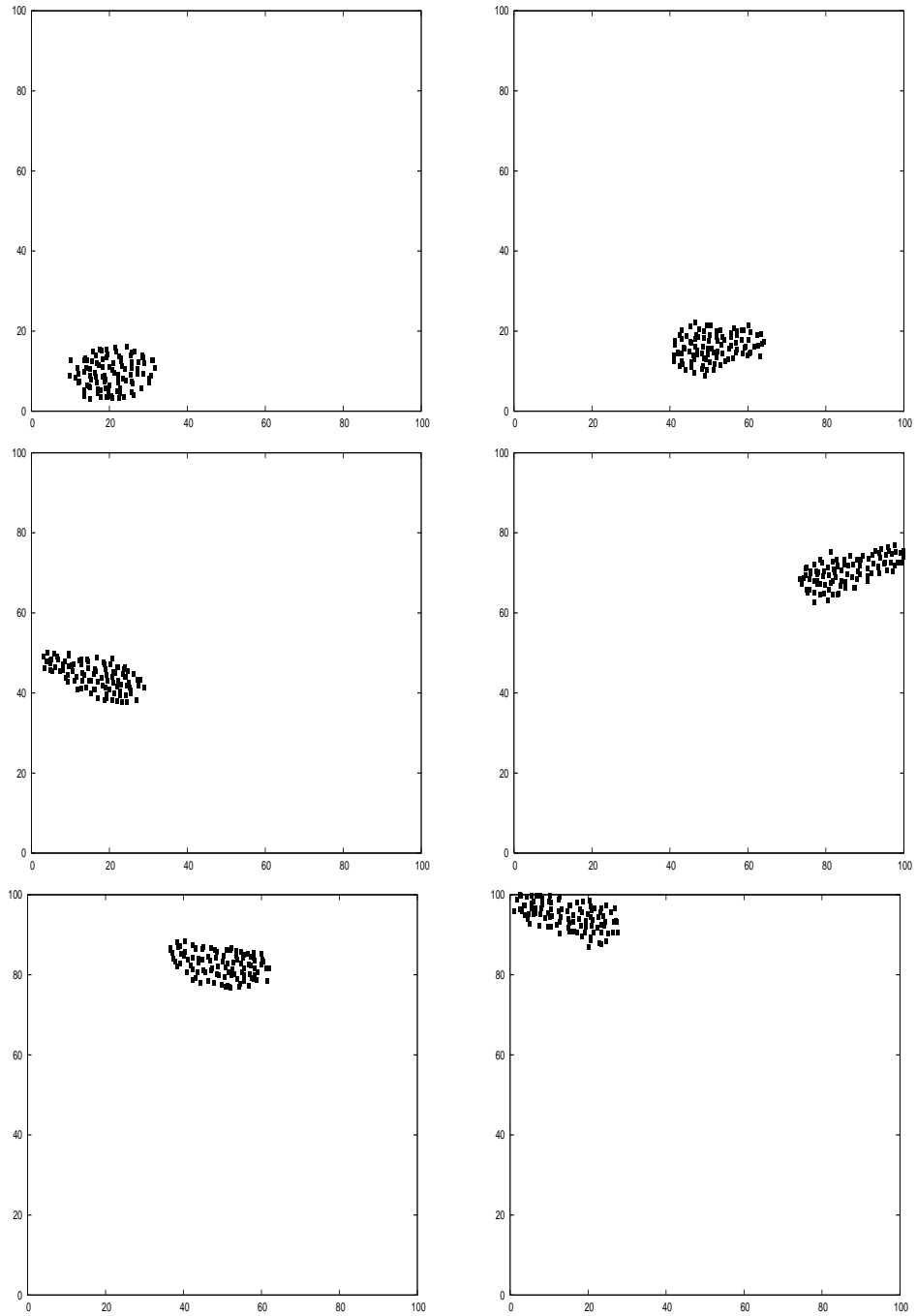


Figure 4.19: Snapshots of GCDL algorithm performing directed motion in Figure 4.16.

with the data gathered within that time frame. We design our algorithms to work with local knowledge only, without the use of any global positioning infrastructure such as GPS, anchor points, and seed nodes. We tested our algorithms with various simulated real-world errors, in both random and directional mobility scenarios and observed that they perform coherent movement even in high noise scenarios.

## Chapter 5

### Conclusion

In this thesis, we first propose a deterministic reservoir sampling algorithm (DRS) designed to sample count data, which is quite common for data mining applications. With extensive simulations we show that DRS generates samples with better accuracy and quality than the previous algorithms.

Later, we investigate sampling as a data reduction method for wireless sensor networks, and propose our Deterministic Weighted Sampling (DWS) algorithm. DWS is a distributed weighted sampling algorithm which is simple enough to not consume too many resources, and distributes the sampling work over the sensor network equally, thus prevents any node from being a bottleneck. DWS is designed to work on arbitrary network topologies, thanks to its weighted strategy. We also presented extensive simulation results on synthetic and real-world datasets to show that DWS generates better quality samples by using far less energy compared to previous sampling algorithms on wireless sensor networks.

As the last part of this thesis, we propose two GPS-free node localization algorithms. Our algorithms work with local knowledge only, without the use of a global positioning infrastructure. The algorithms are also designed to perform localization on demand, in one or two steps of the motion, which makes them memoryless and avoids accumulation of position error over time. We observe that the memoryless property of our algorithms are

most important when coherent movement of the swarm of nodes are required, since without this property accumulation of error affects the localization accuracy and disorganizes the swarm. Furthermore, we designed the algorithms to work under high measurement errors and real world disturbances, excessively tested the algorithms with various simulated real-world errors, and found them robust to the effects of these errors through time.

As a future work of this thesis, it is only natural to extend the data aggregation method we propose here to mobile networks with the help of our localization algorithms. In this new application, a dynamic aggregation tree will be built within the swarm connecting the mobile sink with the rest of the nodes, while the whole swarm is mobile in order to achieve a common goal. The dynamic aggregation tree will allow efficient data flow from within the swarm to the sink as well as aid the communication of the sink node with the rest of the swarm. The key challenge in this approach would be the frequent change in the structure of the aggregation tree due to mobility. Our localization algorithms may help plan the necessary modifications to the aggregation tree in order to keep it connected in mobility scenarios.

## Bibliography

- [1] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Çetintemel, M. Cherniack, J. H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryzkina, N. Tatbul, Y. Xing, and S. B. Zdonik. The design of the Borealis stream processing engine. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR'05)*, pages 277–289, Asilomar, CA, USA, 2005.
- [2] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. *SIGMOD Record*, 22(2):207–216, 1993.
- [3] H. Akcan, A. Astashyn, and H. Brönnimann. Deterministic algorithms for sampling count data. *Data and Knowledge Engineering*, Accepted for publication, 2007.
- [4] H. Akcan, A. Astashyn, H. Brönnimann, and L. Bukhman. Sampling multi-dimensional data. Technical Report TR-CIS-2006-01, CIS Department, Polytechnic University, February 2006.
- [5] H. Akcan and H. Brönnimann. Deterministic data reduction in sensor networks. In *Proceedings of the Third IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS'06)*, pages 530–533, Vancouver, Canada, 2006.
- [6] H. Akcan and H. Brönnimann. A new deterministic data aggregation method for wireless sensor networks. *Elsevier Signal Processing: Special Issue on Information Processing and Data Management in Wireless Sensor Networks*, 87(12):2965–2977, 2007.

- [7] H. Akcan, H. Brönnimann, and R. Marini. Practical and efficient geometric epsilon-approximations. In *18th Canadian Conference on Computational Geometry (CCCG'06)*, pages 121–124, Kingston, Toronto, Canada, 2006.
- [8] H. Akcan, V. Kriakov, H. Brönnimann, and A. Delis. GPS-Free node localization in mobile wireless sensor networks. In *Fifth ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE'06)*, pages 35–42, Chicago, IL, USA, 2006.
- [9] H. Akcan, V. Kriakov, H. Brönnimann, and A. Delis. Facilitating movement of mobile sensors via GPS & Compass free node localization. *Cluster Computing*, Submitted, 2007.
- [10] B. Babcock, M. Datar, and R. Motwani. Sampling from a moving window over streaming data. In *Proceedings of the thirteenth annual ACM-SIAM Symposium on Discrete Algorithms (SODA'02)*, pages 633–634, San Francisco, CA, USA, 2002.
- [11] M. Bădoiu, E. D. Demaine, M. T. Hajiaghayi, and P. Indyk. Low-dimensional embedding with extra information. In *Symposium on Computational Geometry*, pages 320–329, Brooklyn, New York, USA, 2004.
- [12] D. Barbará, W. DuMouchel, C. Faloutsos, P. J. Haas, J. M. Hellerstein, Y. E. Ioannidis, H. V. Jagadish, T. Johnson, R. T. Ng, V. Poosala, K. A. Ross, and K. C. Sevcik. The New Jersey Data Reduction Report. *IEEE Data Eng. Bull.*, 20(4):3–45, 1997.
- [13] B. Bash, J. Byers, and J. Considine. Approximately uniform random sampling in sensor networks. In *First workshop on Data management in Sensor Networks (DMSN'04)*, pages 32–39, Toronto, Canada, 2004.
- [14] P. Bergamo and G. Mazzini. Localization in sensor networks with fading and mobility. In *Personal, Indoor and Mobile Radio Communications*, pages 750–754, Lisbon, Portugal, 2002.



- [15] H. Brönnimann, B. Chen, M. Dash, P. J. Haas, Y. Qiao, and P. Scheuermann. *Efficient data-reduction methods for on-line association rule discovery*. Chapter 4 of Selected papers from the NSF Workshop on Next-Generation Data Mining (NGDM'02), MIT Press, 2004.
- [16] H. Brönnimann, B. Chen, M. Dash, P. J. Haas, and P. Scheuermann. Efficient data reduction with EASE. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 59–68, Washington, DC, USA, 2003.
- [17] N. Bulusu, J. Heidemann, and D. Estrin. GPS-less low cost outdoor localization for very small devices. *IEEE Personal Communications Magazine*, 7(5):28–34, October 2000.
- [18] J. Caffery and G. Stber. Overview of radiolocation in CDMA cellular systems. *IEEE Communications Mag.*, 36(4):38–45, 1998.
- [19] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *Wireless Communications and Mobile Computing*, 2(5):483–502, 2002.
- [20] S. Capkun, M. Hamdi, and J.-P. Hubaux. GPS-free positioning in mobile ad hoc networks. *Cluster Computing*, 5(2):157–167, 2002.
- [21] B. Chazelle. *The discrepancy method*. Cambridge University Press, Cambridge, United Kingdom, 2000.
- [22] B. Chen, P. J. Haas, and P. Scheuermann. A new two-phase sampling based algorithm for discovering association rules. In *Proceedings of the Eighth ACM SIGKDD International Conference*, pages 462–468, Edmonton, Alberta, Canada, 2002.
- [23] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: An energy-efficient coordination algorithm for topology maintenance in Ad Hoc wireless networks. In

*Proceedings of the 7th ACM International Conference on Mobile Computing and Networking (MobiCom'01)*, pages 85–96, Rome, Italy, 2001.

- [24] K. Chintalapudi, R. Govindan, G. Sukhatme, and A. Dhariwal. Ad-Hoc localization using ranging and sectoring. In *INFOCOM*, pages 2662 – 2672, Hong Kong, China, 2004.
- [25] J. Considine, F. Li, G. Kollios, and J. W. Byers. Approximate aggregation techniques for sensor databases. In *Proceedings of the 20th International Conference on Data Engineering (ICDE'04)*, pages 449–460, Boston, MA, USA, 2004.
- [26] J. Considine, F. Li, G. Kollios, and J. W. Byers. Approximate aggregation techniques for sensor databases. In *20th International Conference on Data Engineering (ICDE'04)*, pages 449–460, Boston, MA, USA, 2004.
- [27] M. Datar and S. Muthukrishnan. Estimating rarity and similarity on data stream windows. In *Proceedings of the 10th Annual European Symposium on Algorithms*, pages 323–334, La Sapienza, Italy, 2002.
- [28] A. Deligiannakis and Y. Kotidis. Data reduction techniques in sensor networks. *IEEE Data Eng. Bull.*, 28(1):19–25, 2005.
- [29] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Compressing historical information in sensor networks. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 527–538, Paris, France, 2004.
- [30] N. Duffield, C. Lund, and M. Thorup. Learn more, sample less: Control of volume and variance in network measurements. *IEEE Transactions on Information Theory*, 51(5):1756–1775, 2005.
- [31] P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 331–342, Seattle, WA, USA, 1998.

- [32] P. B. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. In *Proceedings of 23rd International Conference on Very Large Data Bases (VLDB'97)*, pages 466–475, Athens, Greece, 1997.
- [33] P. B. Gibbons, V. Poosala, S. Acharya, Y. Bartal, Y. Matias, S. Muthukrishnan, S. Ramaswamy, and T. Suel. AQUA: System and techniques for approximate query answering. Technical report, Bell Labs, Murray Hill, New Jersey, U.S.A., 1998.
- [34] L. Girod and D. Estrin. Robust range estimation using acoustic and multimodal sensing. In *IEEE/RSI Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 1312–1320, Maui, HI, USA, 2001.
- [35] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data Cube: A relational aggregation operator generalizing Group-By, Cross-Tab, and Sub-Total. In *Proceedings of the Twelfth International Conference on Data Engineering (ICDE'96)*, pages 152–159, New Orleans, LA, USA, 1996.
- [36] M. Greenwald and S. Khanna. Power-conserving computation of order-statistics over sensor networks. In *Proceedings of the Twenty-third ACM Symposium on Principles of Database Systems (PODS'04)*, pages 275–285, Paris, France, 2004.
- [37] S. Guha, N. Koudas, and K. Shim. Approximation and streaming algorithms for histogram construction problems. *ACM Trans. on Database Systems*, 31(1):396–438, 2006.
- [38] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd Hawaii International Conference on System Sciences (HICSS'00)*, Wailea Maui, HI, USA, 2000.
- [39] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. An application specific

- protocol architecture for wireless microsensor networks. *IEEE Trans. of Wireless Communications.*, 1(4):1, 2002.
- [40] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 171–182, Tucson, AZ, USA, 1997.
  - [41] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, Cambridge, MA, USA, 2000.
  - [42] X. Hong, M. Gerla, G. Pei, and C.-C. Chiang. A group mobility model for ad hoc wireless networks. In *International Workshop on Modeling Analysis and Simulation of Wireless and Mobile System (MSWiM'99)*, pages 53–60, Seattle, WA, USA, 1999.
  - [43] L. Hu and D. Evans. Localization for mobile sensor networks. In *The Annual International Conference on Mobile Computing and Networking (MOBICOM'04)*, pages 45–57, Philadelphia, PA, USA, 2004.
  - [44] A. Jain and E. Y. Chang. Adaptive sampling for sensor networks. In *Proceedings of the 1st international workshop on Data management for sensor networks (DMSN'04)*, pages 10–16, Toronto, Canada, 2004.
  - [45] G. H. John and P. Langley. Static versus dynamic sampling for data mining. In *Proceedings of the Second ACM SIGKDD International Conference*, pages 367–370, Portland, OR, USA, 1996.
  - [46] T. Johnson, S. Muthukrishnan, and I. Rozenbaum. Sampling algorithms in a stream operator. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1–12, Baltimore, MD, USA, 2005.
  - [47] Y.-B. Ko and N. H. Vaidya. Location-Aided Routing (LAR) in mobile ad hoc networks. *Wireless Networks*, 6(4):307–321, 2000.

- [48] R. Kohavi, C. E. Brodley, B. Frasca, L. Mason, and Z. Zheng. KDD-Cup 2000 Organizers' Report: Peeling the Onion. *SIGKDD Explorations*, 2(2):86–98, 2000.
- [49] A. Kröller, D. Pfisterer, C. Buschmann, S. P. Fekete, and S. Fischer. Shawn: A new approach to simulating wireless sensor networks. In *Proceedings Design, Analysis, and Simulation of Distributed Systems (DASD'05)*, pages 117–124, Huntsville, AL, USA, 2005.
- [50] V. Kumar and S. R. Das. Performance of dead reckoning-based location service for mobile ad hoc networks. *Wireless Communications and Mobile Computing*, 4(2):189–202, 2004.
- [51] I. Lazaridis and S. Mehrotra. Capturing sensor-generated time series with quality guarantees. In *Proceedings of the 19th International Conference on Data Engineering (ICDE'03)*, page 429, Bangalore, India, 2003.
- [52] K. Levon. Detection of biological warfare agents. In *Proc. BCC Nanotech and Biotech Convergence 2003 Conf.*, pages 169–186, Stamford, CT, USA, 2003.
- [53] S. Lindsey, C. Raghavendra, and K. M. Sivalingam. Data gathering algorithms in sensor networks using energy metrics. *IEEE Trans. on Parallel and Distributed Systems*, 13:924–935, 2002.
- [54] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: A tiny aggregation service for Ad-Hoc sensor networks. In *5th Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, MA, USA, 2002.
- [55] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 491–502, San Diego, CA, USA, 2003.

- [56] A. M. Mainwaring, D. E. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications*, pages 88–97, Atlanta, GA, USA, 2002.
- [57] A. Manjhi, S. Nath, and P. B. Gibbons. Tributaries and deltas: Efficient and robust aggregation in sensor network streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 287–298, Baltimore, MD, USA, 2005.
- [58] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proceedings of 28th International Conference on Very Large Data Bases (VLDB’02)*, pages 346–357, Hong Kong, China, 2002.
- [59] S. Nath, P. B. Gibbons, S. Seshan, and Z. R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys’04)*, pages 250–262, Baltimore, MD, USA, 2004.
- [60] F. Olken and D. Rotem. Random Sampling from Databases - A Survey. *Statistics and Computing*, 5(1):25–42, 1995.
- [61] S. Poduri and G. S. Sukhatme. Constrained coverage for mobile sensor networks. In *IEEE International Conference on Robotics and Automation (ICRA’04)*, pages 165 – 171, New Orleans, LA, USA, 2004.
- [62] N. B. Priyantha, H. Balakrishnan, E. Demaine, and S. Teller. Anchor-free distributed localization in sensor networks. In *1st International Conference on Embedded Networked Sensor Systems (SenSys’03)*, pages 340–341, Los Angeles, CA, USA, 2003.
- [63] M. A. Sharaf, J. Beaver, A. Labrinidis, and P. K. Chrysanthis. Balancing energy effi-

- ciency and quality of aggregate data in sensor networks. *VLDB Journal*, 13(4):384–403, 2004.
- [64] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri. Medians and beyond: new aggregation techniques for sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys'04)*, pages 239–249, Baltimore, MD, USA, 2004.
- [65] I. I. Systems. Synthetic data generation code for associations and sequential patterns. Research group at the IBM Almaden Research Center. <http://www.almaden.ibm.com/software/quest/Resources/datasets/syndata.html>.
- [66] H. Toivonen. Sampling large databases for association rules. In *Proceedings of 22th International Conference on Very Large Data Bases (VLDB'96)*, pages 134–145, Bombay, India, 1996.
- [67] J. S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985.
- [68] Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Record*, 31(3):9–18, 2002.
- [69] Y. Yemini. Some theoretical aspects of position-location problems. In *20th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1–8, San Juan, Puerto Rico, 1979.
- [70] M. J. Zaki, S. Parthasarathy, W. Li, and M. Ogihara. Evaluation of sampling for data mining of association rules. Technical Report TR617, University of Rochester, Rochester, NY, 1996.
- [71] F. Zhao and L. Guibas. *Wireless Sensor Networks: An Information Processing Approach*. Morgan Kaufmann, 2004.