Sensor Ranking: A Primitive for Efficient Content-based Sensor Search

B. Maryam Elahi, Kay Römer, Benedikt Ostermaier
Institute for Pervasive Computing, ETH Zurich
Zurich, Switzerland
elahib@ethz.ch, {roemer, ostermaier}@inf.ethz.ch

Michael Fahrmair, Wolfgang Kellerer
Ubiquitous Networking Research, DOCOMO Communications Laboratories Europe GmbH
Munich, Germany
{fahrmair, kellerer}@docomolab-euro.com

ABSTRACT

The increasing penetration of the real world with embedded and globally networked sensors enables the formation of a *Web of Things* (WoT), where high-level state information derived from sensors is embedded into Web representations of real-world entities (e.g. places, objects). A key service for the WoT is searching for entities which exhibit a certain dynamic state at the time of the query, which is a challenging problem due to the dynamic nature of the sought state information and due to the potentially huge scale of the WoT. In this paper we introduce a primitive called *sensor ranking* to enable efficient search for sensors that have a certain output state at the time of the query. The key idea is to efficiently compute for each sensor an estimate of the probability that it matches the query and process sensors in the order of decreasing probability, such that effort is first spent on sensors that are very likely to actually match the query. Using real data sets, we show that sensor ranking can significantly improve the efficiency of content-based sensor search.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Search process*

General Terms

Design, Algorithms

Keywords

Web of Things, Search, Ranking, Sensors

IPSN'09, April 13–16, 2009, San Francisco, California, USA.

1. INTRODUCTION

The increasing penetration of the real world with embedded and globally networked sensors (e.g., built into mobile phones), enables novel applications that take into account the current state of the real world. This trend has led to the development of middleware (e.g., GSN [4], SenseWeb [14]) that offers unified interfaces to access such sensors. Given these developments, we envision a *Web of Things*, where Web representations of real-world entities (i.e., objects, places, and creatures) are augmented with real-time state information derived from sensors. For example, a Web page that represents a meeting room, containing static information on its size and location, could be augmented with real-time state information on the presence of people in the room. Another example is social networking platforms, where users may share their personal state (e.g., where they are, what they are doing, whom they are with) with their friends by publishing this dynamic state (e.g., derived from mobile phone sensors) on their personal Web pages. As in these examples, we are particularly interested in people-centric sensors whose output is mainly controlled by the behavior of people.

In contrast to existing middleware such as GSN or SenseWeb, the WoT is entity-centric rather than sensor-centric: We believe that users are primarily interested in *entities* of the real-world (e.g., places, objects, creatures) and their high-level *states* (e.g., room is occupied) rather than in *sensors* and their *raw output*.

As in the traditional Web, *search* is also a key service in the WoT, enabling users to find real-world entities that exhibit a certain *current* state (e.g., currently empty rooms of a certain size, locations where many people sharing my interests are currently hanging out). Searching the WoT is a challenging problem as the sought information (i.e., real-time state of entities) is highly dynamic, inherently distributed, and the number of entities in the WoT is potentially huge.

The key problem that needs to be solved to realize a search engine for the WoT is *content-based sensor search*, that is, given a large number of sensors, efficiently find a subset of sensors that output a sought value at a given point in time. In

this paper, we introduce a primitive called *sensor ranking* that enables the implementation of efficient content-based sensor search.

The basic idea is to compute a ranked list of sensors such that the higher the rank of a sensor in this list, the likelier this sensor matches the query. This way, a search engine can process sensors in the order of their rank, spending effort on sensors first that are most likely to match the query. Such effort includes in particular reading the current output value of the sensor over the Internet to check if it actually matches the query. As the latter is a costly operation, sensor ranking helps to avoid spending useless effort on sensors that are unlikely to match the query anyway, and reduces the latency of producing the result pages.

For this approach to be effective, we need to be able to compute the ranking efficiently in the search engine without the need for costly remote operations. To achieve this, we exploit the fact that people are creatures of habit and hence the output of people-centric sensors tends to show periodic behavior [17]. Thus, we can apply a specific class of prediction models that identify such periodic patterns in past output of a sensor and allow to compute an estimate of the probability that the sensor will output the sought value at a future point in time. The ranking is then computed by sorting the sensors by the probability estimates in descending order. Note that with this approach, an inaccurate prediction does not affect the correctness of the query result. Instead, it causes a sensor to be ranked either too high or too low, which causes an increased overhead in the search engine for downloading current values of sensors that do not match eventually.

The idea is that sensor nodes (or gateway computers that relay data streams from sensor nodes to the Internet) autonomously create these prediction models, such that a search engine can occasionally download these prediction models and index them in a local database, much like current Web search engines crawl the Web to download and index static Web pages. This way, computing a ranking in the search engine is a local operation that only accesses the database of indexed prediction models.

To further improve the accuracy of the computed rankings, we introduce a so-called *adjustment process* which measures the overall accuracy of past rankings and adjusts future predictions (independent of the underlying prediction models) such that the accuracy of future rankings is improved.

One may argue that some sensors may not show a periodic behavior, thus rendering our approach inappropriate. However, our experience with different real-world data sets and results published by others [17, 8] indicate that many people-centric sensors do in fact show such a periodic behavior. Hence, among a given set of sensors we can first identify those with a periodic behavior and apply sensor ranking to them. The remaining smaller set of sensors with aperiodic output can be dealt with in a different, more costly way (e.g., frequently reading their current values over the network).

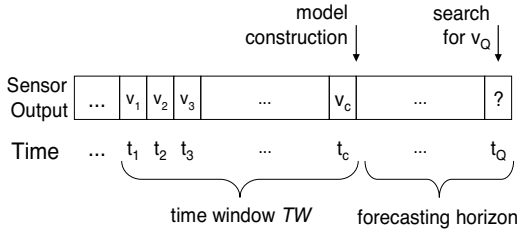

Figure 1: Sensor output time series and the query time.

The remainder of this paper is structured as follows. Sect. 2 introduces a detailed system model, while Sect. 3 presents metrics to measure the accuracy of rankings. We describe different prediction models in Sect. 4. The adjustment process is introduced in Sect. 5. Finally, we evaluate sensor ranking on two real-world data sets in Sect. 6.

2. SYSTEM MODEL

In this section we introduce a formal model of sensor ranking. For this, we assume the existence of sensors that monitor the state of real-world entities. A sensor s is represented by the function

$$\mathrm{s} : \mathcal{T} \mapsto \mathcal{V} \tag{1}$$

where $\mathcal{T}$ denotes discrete time and $\mathcal{V}$ is the finite, discrete set of possible sensor output values representing the high-level state of a real-world entity, e.g., for a sensor monitoring the occupancy of a room $\mathcal{V} = \{\text{“}free\text{”}, \text{“}occupied\text{”}\}$. Note that this notion of a sensor abstracts from the fact that in practice complex processing of low-level sensory data, perhaps from multiple physical sensors, may be required to obtain such high-level state information. However, for the purpose of sensor ranking this is irrelevant and is therefore not reflected in the model.

A prediction model is formally defined as a function

$$\mathrm{P}_{s,t_c,TW} : \mathcal{T} \times \mathcal{V} \mapsto [0,1] \tag{2}$$

meaning that the model has been constructed at time t_c, taking into account the output of sensor s in a time window of size TW prior to t_c as depicted in Fig. 1. We only consider the past sensor readings in a time window TW and not the entire past readings available, since sensor values from the distant past are often poor indicators of what can be expected in the near future. What is more, if the system is going to run for a considerably long time, the resource consumption could prohibit us from using all the past sensor outputs.

Given a point in time $t_Q > t_c$ (the query time) and a sensor value $v_Q \in \mathcal{V}$ (the query value), $\mathrm{P}(t_Q, v_Q)$ returns an estimate of the probability that $\mathrm{s}(t_Q) = v_Q$ holds. We call $h_Q = t_Q - t_c$ the forecasting horizon of the query.

Note that we make on purpose no statement about the type of prediction model used. In fact, every sensor may use a different type of prediction model (e.g., one that gives the best prediction accuracy for that particular sensor).

To perform a content-based sensor search looking for sensors that output value v_Q at time t_Q, the search engine first identifies a subset of sensors R that can possibly match the query. For example, this may be based on static meta information on the type of sought sensors given in the query. However, as the details of this step are not relevant for sensor ranking, it is not reflected in the model.

Next, the ranked list $S^Q = s_1, s_2, \ldots, s_m$ is computed where $s_i \in R$ such that $\forall s_j, s_{j+1} \in S^Q : \mathrm{P}_{s_j}(t_Q, v_Q) \geq \mathrm{P}_{s_{j+1}}(t_Q, v_Q)$.

Finally, sensors in the ranked list are consulted to check whether indeed $\mathrm{s}(t_Q) = v_Q$ holds by reading the current value $s_i(t_Q)$ of sensor s_i over the network starting with $i = 1$ until a sufficient number of matching sensors have been found. To reduce the latency of this step at the cost of increased communication overhead, multiple sensors from the top of the ranked list can be checked in parallel (e.g., using multiple threads).

3. METRICS

To be able to compare the performance of different prediction models, we need to define metrics to measure the accuracy of the rankings computed using the different prediction models.

In an optimal ranking, all sensors that actually match the query would be positioned on top of the list, while all sensors that do not match the query are at the bottom. In a suboptimal ranking, a matching sensor is ranked lower than a non-matching sensor. When the search engine processes the ranked list from top to bottom, reading the current value of a sensor over the network to check if it actually matches the query, then a suboptimal ranking causes extra overhead as the search engine has to consider additional non-matching sensors before it has found enough matching sensors. Hence, to assess the quality of a ranking for a query for value v at time t, we define our basic error metric $e(t, v)$ to measure the overhead for checking non-matching sensors imposed by this ranking. For this, we lookup the last sensor in the ranked list that matches the query and count the number of non-matching sensors that are ranked above this last match. To normalize, we divide this number by the rank of the last match, which gives the ratio of checked sensors that do not match the query.

Let $S^Q = s_1, s_2, \ldots, s_m$ be a ranked list of sensors as defined in Sect. 2. Let L denote the rank of the lowest-ranked sensor that matches the query. We have

$$L = \underset{1 \leq k \leq m}{\arg\max}(\chi(s_k, t, v) * k) \tag{3}$$

where $\chi(s, t, v)$ is an indicator function checking if sensor s outputs v at time t, such that,

$$\chi(s, t, v) = \begin{cases} 1 & : \quad s(t) = v \\ 0 & : \quad \text{else} \end{cases} \tag{4}$$

We formalize the ranking error as follows:

$$e(t, v) = \begin{cases} 0 & : \quad \sum_{i=1}^{m} \chi(s_i, t, v) = 0 \\ \frac{\sum_{i=1}^{L} 1 - \chi(s_i, t, v)}{L} & : \quad \text{else.} \end{cases} \tag{5}$$

Another interesting metric could show us the accuracy of the predictions in terms of the quality of the topmost section of the ranking. Typically, search results are shown to the user in multiple pages, and the probability that user checks each page quickly drops from the first page to the next pages (A study by [13] claims that more than half of the users do not browse the results after the first page). So it is most likely that the search engine needs to find matches only enough to fill the first few pages. Hence a prediction model that results in a ranking in which the top segment of the ranked list has less non-matching sensors, in practice is better than one that results in a ranking with the same ranking error, but with a less accurate top segment.

We define the top-m' ranking error as the ratio of mismatches in the first m' entries of the ranked list, i.e.

$$e_{top}(t, v, m') = \begin{cases} 0 : \sum_{i=1}^{m} \chi(s_i, t, v) = 0 \\ \frac{\sum_{i=1}^{\min(L, m')} 1 - \chi(s_i, t, v)}{\min(L, m')} : \text{else.} \end{cases} \tag{6}$$

4. PREDICTION MODELS

To implement sensor ranking, we need prediction models to compute estimates of the probability that a given sensor generates a given output value at a given point in time. To be applicable for sensor ranking, prediction models have to meet a number of requirements. Firstly, they need to operate on categorical time series as the codomain $\mathcal{V}$ of sensors in our system model is a discrete set of nominal values without any inherent ordering. Secondly, given that existing Web search engines crawl the Web with a frequency of days to weeks, we cannot assume a search engine for the WoT to be able to crawl the WoT to download and re-index prediction models much faster than that. Hence, prediction models should be able to produce accurate probability estimates for a forecasting horizon in the order of days to weeks.

Our approach to prediction exploits the fact that many people-centric sensors tend to show a periodic behavior [17, 8]. However, this periodic behavior is far from being perfect (e.g., period lengths change over time). Therefore, thirdly, prediction models need to be able to deal with such imperfect periodic behavior.

In this section we first discuss the characteristics of the output of people-centric sensors and the resulting challenges for prediction models. Then, we present two specific prediction models.

4.1 Challenges

Although the output of people-centric sensors usually reflects on the periodic nature of people's behavior and thus

follows periodic patterns, *perfect* periodic patterns are rarely observable in these outputs.

The output of many sensors that monitor real-world entities is often affected by multiple real-world processes with different period lengths. For example a meeting room could host weekly group meetings, and daily briefings. The state of some entities might only be affected by one person, e.g., the occupancy of an office room is usually affected by the behavior of the owner of the office, but the behavior of a person is also a composition of several intermittent periodic processes. For example, every day the person goes out for dinner and lunch. Every week she could go to meetings on Monday and she does not work during the weekends.

When we have multiple periodic processes in the time series of our sensor outputs, they can overlap at some points and even have conflicts. For example, a person leaves her office for lunch every day at 1pm, but on Tuesdays she has lunch with her friends, therefore she leaves the office at 2pm. So the office is empty almost every day at 1pm, but not on Tuesdays. Deviations from plans and schedules are very common for people, so exceptions in the periodic behavior are also frequent in the output of people-centric sensors. What is more, periodic patterns usually change over time and are replaced by new patterns. For example, the occupancy pattern for a classroom changes at the beginning of each semester.

So the output of people-centric sensors are often not a composition of perfectly periodic patterns, but rather a combination of overlapping, semi-periodic, and partially periodic patterns with exceptions and noise. We call a state semi-periodic with regard to a period length, if it reoccurs almost every period with exceptions from time to time. A partially periodic pattern is a pattern in which some elements reoccur with the same period length, while the other elements does not repeat with regard to that period length. For example, the occupancy of an office room has a partial periodic pattern with period length of a day, in which the state of the room is always empty during the nights, but the room could have different states in the morning from day to day.

To make accurate predictions for such sensor outputs, we need to detect periodic patterns that are indicative of an actual real-world periodic process. Furthermore we need to detect changes in the underlying periodic processes over time and decide what is the dominant state when patterns overlap, and possibly cope with exceptions and noise as well.

This proves to be a hard problem since the future of high-entropy sensor outputs is not easily predictable [8]. We say a sensor output has high entropy if it is composed of many overlapping periodic processes with frequent exceptions and changes over time.

4.2 Single-Period Model

If we assume that there is a dominant period with length p, after which the sensor output is likely to repeat, we could expect different points in time with the same phase (offset in the period) to have similar values. For example, we would expect the occupancy of a class room to follow almost the same pattern every week and the occupancy of a class room is likely to be different on Mondays than on Sundays. Assuming that the time window TW is an integral multiple of p, and that the forecasting horizon is smaller than p, we consider the following single-period prediction model[1]:

$$\mathrm{P}_{s,t_c,TW}(t,v) = \frac{\sum_{i=1}^{TW/p} \chi(s, t - ip, v)}{TW/p} \tag{7}$$

With this approach we compute the probability of a state v at time t by considering all points t_i in the time series that have the same phase as t with respect to period length p. In other words, the prediction model computes the fraction of the instances of t_i for which $s(t_i) = v$, for all $t_i \in [t_1, t_c]$, where $t_i \equiv t \bmod p$.

To determine the dominant period length, we can either use domain knowledge, or we can automatically extract candidate periods by spectral analysis of the sensor output values by methods such as those presented in [6, 12].

This model is both simple and computationally efficient, however, it ignores the fact that sensor output is usually composed of many periodic processes with different period lengths.

4.3 Multi-Period Model

The state of many real world entities is often affected by different periodic activities, so sensor outputs are usually the collective result of several periodic processes with different period lengths.

Below we introduce a multi-period prediction model which takes into account that multiple periodic real-world processes with different period lengths affect the output of a people-centric sensor. The model consists of two major steps. In the first step, we discover so-called *periodic symbols* in a time window of past sensor output. Each such periodic symbol is a tuple (v, p, l, ϕ), where v is a sensor output, p is a period length, l is a phase (i.e., offset in a period), and $0 \leq \phi \leq 1$ is the *support* of the periodic symbol in the input time series. For example, the periodic symbol ("occupied", 7 days, day 4, 0.8) means that the sensor output "occupied" repeats every week (period of 7 days) on Friday (fifth day in the week starting from zero) with a support of 0.8.

In the second step, we use the discovered set of periodic symbols to make a prediction at time t_Q. For this, we filter periodic symbols that map to the query time, i.e., $t_Q \equiv l \bmod p$. Using the support values of the remaining periodic symbols, we compute the probability estimate, i.e., the output of the prediction model.

[1]Note that the model can be easily extended to work with any period length and forecasting horizon, but the resulting formula is somewhat more complex and obfuscates the underlying principle, so we show this simplified version here.

The algorithm to efficiently discover periodic symbols is based on previous work by Elfeki [9]. However, as this algorithm wasn't originally intended for making predictions, we introduce several modifications to the algorithm. We first summarize the original algorithm and then describe our modifications. Finally, we describe how the resulting set of periodic symbols is used to make predictions.

4.3.1 Discovery of Periodic Symbols

To make the paper self-contained, we summarize the algorithm for discovery of periodic symbols presented in [9] and point out where we modified their algorithm. The actual modifications are detailed in the subsequent sections.

The input of the algorithm consists of the contents of the time window $V = v_0, v_1, \ldots, v_{n-1}$, where $v_i \in \mathcal{V} = \{\alpha_0, \alpha_1, ..., \alpha_{\sigma-1}\}$. The value of an element v_i in this time series is called a *symbol* over the alphabet $\mathcal{V}$. We borrow the notion of *symbol periodicity* from [9] and call α_k a periodic symbol with period p, if it appears roughly every p timestamps in the time series. We say roughly and not exactly every p time units, since we assume that V is a composition of imperfect patterns as discussed in Sect. 4.1. This assumption is due to the fact that although human-centric activities and states tend to follow repetitive patterns, they are very much prone to exceptions and temporary or permanent changes.

For every state v_i of the time series V we like to find the possible period lengths with which v_i reoccurs in V. For example, in a time series of room occupancy, if we have the state "occupied" for a Monday, we like to know if this state is part of a periodic weekly, two-weekly, etc. pattern, i.e., we want to compute the probability of the room being "occupied" every Monday, every other Monday, every third Monday, etc. In other words, if state v_i at time t_i in the time series V is equal to symbol α_k, we want to compute the probability or the *support* for α_k reoccurring with regard to different period lengths p. Note that for each period length p, timestamp t_i maps to a different offset l in the period, such that $l = t_i \bmod p$.

We define *symbol periodicity support* $\mathcal{SPS}$ as a function

$$\mathcal{SPS} : \mathcal{V} \times \mathcal{L} \times \mathcal{O} \mapsto [0, 1] \tag{8}$$

where $\mathcal{V}$ is the set of possible symbols, $\mathcal{L} = \{1, 2, ..., n/2\}$ is the set of possible period lengths p, and $\mathcal{O} = \{0, 1, ..., p-1\}$ is the set of possible offsets in the period.

A primitive way of finding *symbol periodicity support* is to take a statistical approach. We can iterate over all possible periods p and for each offset l in the period compute the ratio of appearances of symbol α_k. To explain this, let us denote the projection of a time series V according to a period p starting from position l with $\pi_{p,l}(V)$ [9]. We have

$$\pi_{p,l}(V) = v_l, v_{l+p}, v_{l+2p}, \ldots, v_{l+mp} \tag{9}$$

where $l < p, m = \lceil (n-l)/p \rceil$.

For a time series $V = \alpha_1\alpha_2\alpha_3\alpha_1\alpha_2\alpha_1\alpha_3\alpha_2\alpha_1\alpha_1\alpha_2\alpha_3\alpha_1$, $\pi_{3,1}(V) = \alpha_2\alpha_2\alpha_2\alpha_2$ and $\pi_{3,0}(V) = \alpha_1\alpha_1\alpha_3\alpha_1\alpha_1$. So we say the probability or the support for α_2 being periodic with period 3 at the second position is $\mathcal{SPS}(\alpha_2, 3, 1) = 1$ and the support for α_1 being periodic with period 3 at the first offset is $\mathcal{SPS}(\alpha_1, 3, 0) = 4/5$.

A more elaborate approach is introduced by [9] to compute the support for symbol periodicity that considers only consecutive appearances of a symbol in a projection. This allows for a stronger support for close to perfect periodic symbols and a weaker support for patterns with frequent intermittent exceptions and outliers. The number of consecutive appearances of a symbol α_k in a projection series $\pi_{p,l}(V)$ is denoted as $\mathcal{F}_2(\alpha_k, \pi_{p,l}(V))$. Based on this approach, we define our *symbol periodicity support* ϕ as follows.

DEFINITION 1. *In a time series V of length n, for each symbol α_k at offset l of period p, the* symbol periodicity support *is* $\phi = \mathcal{SPS}(\alpha_k, p, l) = \frac{\mathcal{F}_2(\alpha_k, \pi_{p,l}(V))}{\lceil (n-l)/p \rceil - 1}$.

With this definition, for the time series of the above example, $\mathcal{SPS}(\alpha_1, 3, 0) = 2/4$ and $\mathcal{SPS}(\alpha_2, 3, 1) = 1$.

We refer to the set of periodic symbols as $\mathcal{PS}$ where each periodic symbol $ps_i \in \mathcal{PS}$ is a quadruple of the form (α_k, p, l, ϕ). In [9], the set $\mathcal{PS}$ is considered to contain only periodic symbols whose *symbol periodicity support* ϕ is greater than a periodicity threshold. This is to prune weak and possibly useless periodic symbols. However, for our purpose which is building a prediction model, for every possible query time t_Q and for every possible query value v_Q, we need to have at least one periodic symbol that corresponds to the queried value and maps to the query time. Therefore, periodic symbols with low supports are not considered useless in our case, although some of the periodic symbols with low or medium supports could have undesirable effects on the prediction phase. So we devise a more elaborate pruning mechanism, which we discuss later when we introduce *period dominance* in Sect. 4.3.3.

To find a periodic symbol $ps_i = (\alpha_k, p, l, \phi)$, we need to compute $\mathcal{F}_2$, the number of consecutive occurrences of the symbol in the projection $\pi_{p,l}(V)$ of the time series V. With an approach somewhat similar to autocorrelation, [9] shows that we can compute $\mathcal{F}_2(\alpha_k, \pi_{p,l}(V))$ by comparing V with $V^{(p)}$ at positions t_j in the time series, where $V^{(p)}$ is V shifted p positions to the right and $l = t_j \bmod p$. By comparing V to $V^{(p)}$, we are comparing every symbol at timestamp t_j with the symbol at the timestamp t_{j+p}, thus we can detect consecutive occurrences of a symbol. To mine all periodic symbols, we need to shift and compare the time series for all possible periods. Introducing a proper mapping scheme for the symbols in the time series, [9] uses the concept of convolution to efficiently shift and compare the time series for all possible periods. We briefly go over the concept of convolution and describe the modified algorithm used to mine periodic symbols.

Let $X = x_0, x_1, \ldots, x_{n-1}$ and let $X' = x'_0, x'_1, \ldots, x'_{n-1}$

be two sequences of length n. The convolution of X and X' is defined as a sequence $(X \otimes X')$ of length n such that $(X \otimes X')_i = \sum_{j=0}^{i} x_j x'_{i-j}$. Assume $x'_i = x_{n-1-i}$, i.e., X' is the reverse of X. We define sequence C^X as the reverse of convolution of X and X', i.e. $C^X = (X \otimes X')'$. In such a sequence, component c_i of C^X corresponds to positioning X over $X^{(i)}$ and "comparing" corresponding symbols, i.e., $c_i^X = (X \otimes X')'_i = \sum_{j=0}^{n-1-i} x_{n-1-j} \cdot x_{n-1-i-j}$.

Back to our goal which is mining periodic symbols, we want to count the number of consecutive occurrences of a symbol in time series V with regard to period p and position l, by comparing V and $V^{(p)}$. We showed that we can compare V with the corresponding elements in the shifted series $V^{(p)}$ for all possible values of p by computing the convolution C^V. For comparison of V and $V^{(p)}$ we need a mechanism to figure out which symbols are matching at what position in the period p. To locate matches, [9] devises a mapping $\Phi(V)$ such that only matching symbols in V and $V^{(p)}$ contribute to $c_i^V = \sum_{j=0}^{n-1-i} \Phi(v_{n-1-j}) \cdot \Phi(v_{n-1-i-j})$, i.e., $\Phi(v_{n-1-j}) \cdot \Phi(v_{n-1-i-j})$ contributes 1 if symbols v_{n-1-j} and $v_{n-1-i-j}$ are the same and 0 otherwise.

To map symbols, [9] uses the binary representation of increasing powers of two [5], such that α_k is mapped to the binary presentation of 2^k using σ bits. That is, $\bar{V} = \Phi(V)$ is a sequence of zeros and ones of length σn, where $\bar{v}_j = 1$ if $v_{(j/\sigma)} = \alpha_{(j \bmod \sigma)}$. Applying convolution on $\bar{V}$, a sequence $C^{\bar{V}}$ of length σn is constructed in a such way that $c_i^V = c_{\sigma i}^{\bar{V}}$. This is equivalent to using the projection function (9) on $\bar{V}$, i.e., $C^V = \pi_{\sigma,0}(\bar{V})$.

With this mapping, we can determine the number of matches when V is compared to $V^{(i)}$, however, it does not reveal the matching symbols and their positions in the period. To address this problem, a subtle modification is applied to the definition of convolution by [9]. The modified convolution is defined such that $(X \otimes X')_i = \sum_{j=0}^{i} 2^j x_j x'_{i-j}$. Coefficient 2^j contributes a distinct value for each match, based on the position of the match and the matching symbol. Next, we will explain how the identity of the matching symbols and their positions are computed.

For time series V of length n, let $\mathcal{V} = \alpha_0, \alpha_1, \ldots, \alpha_{\sigma-1}$ denote the set of its possible symbols. $\bar{V}$ is constructed by mapping each symbol α_k to the σ-bit binary representation of 2^k. The modified convolution sequence $C^{\bar{V}}$ is composed of components [2] $c_i^{\bar{V}} = \sum_{j=0}^{n-1-i} 2^j \cdot \bar{v}_{n-1-j} \cdot \bar{v}_{n-1-i-j}$ for $i = 0, 1, \ldots, \sigma n - 1$. The sequence C^V is computed using the projection function represented in equation (9), such that $C^V = \pi_{\sigma,0}(C^{\bar{V}})$.

To find the matching symbols and their positions for period p, we extract W_p, the set of powers of 2 that indicate the matches when V is compared to $V^{(p)}$. For non-zero components of C^V, $W_p = \{w_{p,1}, w_{p,2}, \ldots\}$ denotes the set of powers of 2 such that $c_p^V = \sum_h 2^{w_{p,h}}$.

Each power of 2 in W_p is contributed by a pair of matching symbols when comparing V to its shifted version $V^{(p)}$, so the cardinality of W_p represent the number of matching symbols for the period p. This could be an indicator to show whether period p is a dominant period in time series V or not. We exploit this later for pruning in Sect. 4.3.3.

For a period length p, to find symbols that occur in at least two consecutive periods at offset l of the period and to compute the value of the symbol periodicity support $\mathcal{SPS}(\alpha_k, p, l)$, we first extract the set of powers of two that corresponds to symbol α_k, i.e.

$$W_{p,k} = \{w_{p,h} : w_{p,h} \in W_p \wedge w_{p,h} \bmod \sigma = k\} \quad (10)$$

Then for each offset l in period p, we find a subset of $W_{p,k}$ that maps to position l of the period p;

$$W_{p,k,l} = \{w_{p,h} : w_{p,h} \in W_{p,k} \wedge (n - p - 1 - \lfloor w_{p,h}/\sigma \rfloor) \bmod p = l\} \quad (11)$$

The cardinality of $W_{p,k,l}$ is equal to the number of consecutive occurrences of symbol α_k at offset l of period p, in other words, $|W_{p,k,l}| = \mathcal{F}_2(\alpha_k, \pi_{p,l}(V))$. Having $\mathcal{F}_2$ we can compute the symbol periodicity support $\phi = \mathcal{SPS}(\alpha_k, p, l)$ based on definition (1).

4.3.2 Time Window Selection

To compute the symbol periodicity support for all period lengths, the original algorithm in [9] considers the entire time series, thus supports for small periods are computed considering many instances of the period in the past while for large period lengths a small number of period instances are available. This has undesirable consequences on the accuracy of the computed periodicity supports and reduces the comparability of these support values. Hence, we introduce a new approach for selecting the time window from which to discover periodic symbols.

When considering too few instances for computing the supports for periodicity and dominance, we mine periodic symbols with too high or too low supports, without enough supporting evidence. That is because for the few instances available, some symbols happen to shape a pattern and for some symbols who actually follow a close to perfect pattern, the presence of noise or exceptions could undermine this periodic behavior to a great extent. For example if we are only considering three instances for a large period length, if a symbol happens to reoccur just twice in these instances, it would have a relatively high support of 2/3, while a close to perfectly periodic symbol, which happens not to occur in the middle of the three instances we are considering due to an exception, would cause the support to become 0. This problem is more visible when we have a small number of possible symbols. To alleviate this problem, we define a lower

[2] In [9], to compute c_i^V the upper limit of the summation and the timestamp indexes are incorrectly set similar to the original definition of convolution, while C^V is defined to be the reverse of the convolution of V with its reverse. The correct indexes and limit is presented here.

limit ω_{min} for the minimum number of period instances that should be taken into account for mining periodic symbols. We enforce this lower limit by limiting the maximum length of a period to n/ω_{min}.

Considering too many period instances also reduces the accuracy of the support. This is due to the fact that periodic behavior may change over time or a periodic symbol may seize reappearing from a certain point in the time series. For example, if a class room is occupied every Monday morning of every week for a Calculus I class in the winter semester, it could become free for almost all the spring semester when there are no classes held there. If we consider too many instances for a period, we run the risk of incorporating the effects of an old periodic pattern that has changed over the course of time into another pattern. So we impose an upper bound ω_{max} on the number of period instances that are taken into account for computing the support for periodic symbols and period dominances. For this, we modify the upper bounds in the convolution formula, such that for each period length p, $i = \sigma p$, $c_i^{\bar{V}} = \sum_{j=0}^{u_i} 2^j \cdot \bar{v}_{n-1-j} \cdot \bar{v}_{n-1-i-j}$, where the upper bound $u_i = \min(\sigma n - 1 - i, i \cdot \omega_{max} - 1)$.

4.3.3 Pruning and Weighting

With the help of the modified convolution we compute the power set W_p for all possible periods, and consequently obtain the set of periodic symbols $\mathcal{PS}$. Since we did not prune these periodic symbols with a symbol periodicity support threshold as done in [9], all possible periodic symbols are included in $\mathcal{PS}$. However, we are only interested in indicative periodic symbols. We call a periodic symbol indicative, if it reflects the existence of a real-world periodic process in the time series. For example, if there is a class scheduled every Monday morning in a classroom, we obtain a periodic symbol for state "occupied" with a support close to 1 and a periodic symbol for state "free" with support close to 0 (assuming there could be exceptions like holidays). These periodic symbols are considered indicative.

For many sensor output time series, it is often not straightforward to distinguish between indicative and non-indicative periodic symbols in $\mathcal{PS}$, since many of the periodic symbols have irrelevant period lengths, yet they gain relatively high support due to the abundance of a symbol in the time series. We call a period length p irrelevant, if a periodic pattern with period p exists in the time series, but there is no actual real-world periodic process with that period length. For example, in the output time series of a sensor monitoring a class room, state "free" is abundantly present, because of the night hours, the weekends, the holidays, and several hours during the day when no activities are scheduled in this room. Therefore many periodic symbols for state "free", with irrelevant period lengths such as 5 hours, obtain a relatively high periodicity support of $\phi \geq 0.5$. Irrelevant and non-indicative periodic symbols that have supports greater than 0.5 can jeopardize the effect of indicative periodic symbols in the prediction phase, so we need to filter them out of $\mathcal{PS}$.

One approach for pruning non-indicative periodic symbols is to define a set of relevant periods for the model, with which periodic symbols with irrelevant period lengths are filtered out. We can use domain knowledge to introduce relevant period lengths to the model. For example, if we know that the dominant periodic patterns for a library study hall have daily and weekly period lengths, it is reasonable to only look for periodic symbols with these periods.

If we do not have the domain knowledge about the behavior of the sensor output in advance, or if there is a possibility for periodic patterns with different period lengths to appear and replace each other during the course of time, we need to mine dominant periods automatically. To determine the dominance of a period, we use the ratio of the symbols that reoccur with respect to this period length. As mentioned before, cardinality of W_p represent the number of symbol reoccurrences with respect to period p, so with no extra computational cost, we can determine the period dominance for all possible periods in the time series.

DEFINITION 2. *In a time series V of length n, for each period of length p, if $W_p = \{w_{p,1}, w_{p,2}, \ldots\}$ denotes the set of powers of 2 such that $c_p^V = \sum_h 2^{w_{p,h}}$, the* period dominance *is defined as $\mathcal{PD}(p) = \frac{|W_p|}{\min(n-p, p \cdot \omega_{max})}$. If $\mathcal{PD}(p) \geq \delta$, period p is called a* dominant period *with respect to a period dominance threshold $0 < \delta \leq 1$.*

Based on the entropy of the sensor output, a period dominance threshold δ should be chosen carefully in such a way to both prune periodic symbols with irrelevant period lengths and to keep indicative periodic symbols that belong to a less dominant period. A less dominant period is one that shows strong periodicity for some offsets and has arbitrary behavior for the rest of the positions in the period. This results in periodic symbols which are non-indicative but whose period dominance is greater than the threshold to pass the pruning. To alleviate the presence of non-indicative periodic symbols with high supports, we weight the symbol periodicity supports with the dominance of their period, i.e. the output of our algorithm is a set of periodic symbols $\mathcal{PS}'$, which is pruned and weighted by period dominances, such that $\mathcal{PS}' = \{(\alpha_k, p, l, \phi') | (\alpha, p, l, \phi) \in \mathcal{PS} \wedge \mathcal{PD}(p) \geq \delta \wedge \phi' = \phi \cdot \mathcal{PD}(p)\}$.

4.3.4 Inferring Prediction Estimates

To build the prediction model, first we run the modified periodic symbol mining algorithm on the input time series V, which outputs a set $\mathcal{PS}'$ of quadruples $ps'_i = (\alpha_k, p, l, \phi')$. As explained in the previous section, ps'_i means that the probability estimate for symbol α_k reoccurring with period p at offset l is equal to ϕ'. For example, if we have a sensor monitoring the occupancy of a room with a time series resolution of one day, and assuming t_0 is a Monday, $ps'_i = (\text{`}f', 7, 5, 0.8)$ means that state 'f' or "free" reoccurs

every Saturday ($l =$ day 5) of every week ($p = 7$ days) with a support equal to 0.8.

We want to infer predictions for the future sensor output based on $\mathcal{PS}'$, the set of mined periodic symbols. For a query at time $t_Q \geq t_n$, let $\mathcal{PS}'_{t_Q}$ denote the set of periodic symbols that map to time t_Q, i.e. $\mathcal{PS}'_{t_Q} = \{(\alpha_k, p, l, \phi') \in \mathcal{PS}' | t_Q \bmod p = l\}$. This is the set of periodic symbols that could potentially influence the state of the sensor at time t_Q. To make a prediction for the sensor output matching the sought value v_Q at time t_Q, we use the periodic symbol that has the maximum support ϕ' among $\mathcal{PS}'_{t_Q}$ as this periodic symbol is most likely to determine the output of the sensor at the time of the query, i.e.

$$\mathrm{P}(t_Q, v_Q) = \max\{\phi'_i | (\alpha_k, p, l, \phi'_i) \in \mathcal{PS}'_{t_Q} \wedge \alpha_k = v_Q\} \tag{12}$$

For sensors that can assume only two states α_0 or α_1, to estimate the probability of a state v_Q at a time t_Q, it could be reasonable to assume that the state of the sensor at time t_Q is most likely determined by the periodic symbol with the maximum support among those that map to time t_Q, regardless of the value of this symbol. If the value of this periodic symbol matches the query value v_Q, the prediction estimate is equal to its support ϕ', otherwise, it is equal to $1 - \phi'$. This is only reasonable for sensors with two possible states, where the periodic symbol with the maximum support could be a good indicator for estimating the probability of both states, i.e., a strong support ϕ' for state α_1 occurring at a point in time, directly implies a weak probability equivalent to $1 - \phi'$ for the other state α_0 occurring at the same time. Obviously, this is not applicable to sensors with more than two possible states, because a strong ϕ' for state α_1 only implies that the probability for state α_0 is less than $1 - \phi'$.

The question arises, whether the maximum probability estimates for all possible sensor states occurring at time t_Q sum up to 1. The answer is no, due to the fact that sensor output is comprised of multiple periodic processes that overlap and conflict at points in time. For example, a biology group hold their weekly meetings every Friday. They also take field trips every first day of each month without any exception. So for a Friday that is also the first day of a month, we have two periodic symbols: one for state "occupied" every week with a strong support $\phi'_o > 0.5$, and a perfectly periodic symbol with support $\phi'_f = 1$ for state "free" on the first day of every month. It is obvious that in such a case for a query for the state "occupied", it is preferable to return the probability estimate as $1 - \phi'_f = 0$ rather than ϕ'_o. For sensors monitoring low-entropy subjects with non-conflicting periodic processes, the sum of supports, however, is close to one and we can safely return the maximum support of the periodic symbol with the queried value as the probability estimate.

The complete algorithm for constructing a prediction model has computational complexity O(n^3), where n is the size of the time window. However, by using additional space in the order of O(n) for storing intermediate results, we can reduce the computational complexity to O(n^2).

5. ADJUSTMENT PROCESS

Using the prediction models introduced in the previous section, a search engine would compute a rank list S^Q containing sensors s_i sorted by decreasing probability of matching a query for sensor value v_Q at time t_Q. In the best case, all sensors actually matching the query are at the top of the rank list, while all non-matching sensors are at the bottom. Imperfect rankings result if a sensor s_i is misranked due to one of the following reasons. Firstly, s_i matches the query but is ranked lower than other sensors that do not match the query. Secondly, s_i does not match the query but is ranked higher than other sensors that match the query. Formally, we can measure this ranking error for sensor s_i by counting the number of non-matching sensors ranked higher than a matching sensor s_i, respectively by counting the matching sensors ranked lower than a non-matching sensor s_i. The sign of the following metric indicates if s_i should have been ranked lower (< 0) or higher (> 0).

$$\mathrm{re}(s_i, v_Q, t_Q) = \tag{13}$$
$$\begin{cases} -|\{s_j \in S^Q | j > i \wedge s_j(t_Q) = v_Q\}| : s_i(t_Q) \neq v_Q \\ \quad |\{s_j \in S^Q | j < i \wedge s_j(t_Q) \neq v_Q\}| : s_i(t_Q) = v_Q \end{cases}$$

During experiments with our system on realistic data sets we observed that sudden changes in the periodic processes underlying the data (e.g., end of a semester where room occupancy patterns in a university change drastically) typically causes sensors to be consistently misranked until the old data is shifted out of the time window, resulting in a decreased performance of the search process.

To deal with this problem, we introduce an adjustment process that modifies the probabilities computed by the prediction models using the above-defined ranking error $\mathrm{re}()$ such that systematic misranking is corrected. Essentially, we introduce a control loop, where the ranking error of s_i controls the ranking of s_i in future queries such that the future ranking error is reduced. Formally, we compute an adjustment term $\mathcal{AT}$ for each sensor s_i and sought value v_Q at query time t_{Q_j}:

$$\mathcal{AT}(s_i, v_Q, t_{Q_j}) = \mathcal{AT}(s_i, v_Q, t_{Q_{j-1}}) + \frac{\mathrm{re}(s_i, v_Q, t_{Q_j})}{m} \tag{14}$$

where the size m of the rank list S^Q is used to normalize the ranking error. The probability estimate for a sensor s_i holding a state v_Q at the time $t_{Q_{j+1}}$ of the subsequent query is then modified using the above adjustment term:

$$\hat{\mathrm{P}}_{s_i}(t_{Q_{j+1}}, v_Q) = \mathrm{P}_{s_i}(t_{Q_{j+1}}, v_Q) + \mathcal{AT}(s_i, v_Q, t_{Q_j}) \tag{15}$$

The rank list resulting from these adjusted probabilities is then in turn used to update the adjustment terms according to (14). To avoid the use of outdated adjustment values, $\mathcal{AT}$

is reset to zero when a sensor hasn't been queried for some time or when the prediction model of the sensor is updated.

The adjustment process is most effective if the query is executed frequently. However, if the query is executed infrequently, then its contribution to the overall performance of the search engine will be small anyway. Note that the above approach can be used with any prediction model.

6. EXPERIMENTAL RESULTS

In this section we evaluate the performance of the prediction models with respect to the resulting ranking accuracy. First we introduce our evaluation method, then we describe the data sets that have been used for the evaluation, next we specify the model parameters that have been used during the evaluation, and finally we present the evaluation results.

6.1 Evaluation Method

To evaluate the accuracy of sensor ranking for the prediction models introduced in Sect. 4, we simulate the behavior of a search engine in a C++ program using realistic data sets.

Recall from Sect. 2 that sensor nodes (or gateway computers) autonomously construct prediction models and the search engine periodically downloads and indexes these models. The search engine then uses these indexed models to compute rankings in response to queries with a maximum forecasting horizon h before new models are downloaded, indexed, and used to compute rankings.

In our evaluation we consider an interval $[t_b, t_e]$ from the data sets. At times t_i with $t_0 = t_b$ and $t_{i+1} = t_i + h$, models are constructed for every sensor using sensor output in the time interval $[t_i - TW, t_i]$. Here, h, is the maximum forecasting horizon and TW the size of the time window. For each point in time $t_j \in [t_i, t_i + h]$, a query is posed to the simulated search engine for all possible sensor values in $\mathcal{V}$. In response to such a query, the sensor ranking is computed using the current prediction models. The metrics introduced in Sect. 3 are then applied to the ranking to compute the ranking errors. Note that this approach is deterministic, that is, each run of the experiment will produce the same results.

We then compute average ranking errors over all query instances for the two metrics introduced in Sect. 3. In addition, we compute an average ranking error as a function of the forecasting horizon q:

$$\bar{e}_{mod}(q, v) = \frac{h}{t_e - t_b} \sum_{i=0}^{(t_e - t_b)/h - 1} e(t_b + i \cdot h + q, v) \quad (16)$$

If the maximum forecasting horizon h is properly selected (e.g., one week), then this gives us average ranking errors for certain times of the day on certain days of the week (e.g., Monday 10 am).

6.2 Data Sets

We use two datasets of recorded traces of room occupancy data to evaluate our models. The first one, ETH, is extracted from the logs of our university's online room reservation system [10], and the the second one is extracted from the MERL motion detector data set [22].

The online reservation system of ETH has recorded the room reservations for seven seminar rooms starting from the year 2002. To produce our ETH data set, we have downloaded the reservation states of these rooms and turned them into a time series of room occupancy. We assumed if a seminar room was marked as reserved, it had the state "occupied" and if there was no reservation made, it had the state "free". The reservation system only considers the hours between 7am and 10pm, so we have inserted the state "free" for the remaining night hours. The timestamp resolution in this dataset is 1 hour.

The MERL data set [22] contains the output of more than 200 passive infrared (PIR) motion sensors deployed in an office building. Each motion detection in this data set is a instantaneous timestamped event indicating when motion has been detected. We applied a simple filter to this data set to deduce the occupancy status of the surrounding space of the sensors. With this filter, a space is "occupied" if there have been at least two movement events within the past seven minutes, such that there are at least two minutes between the events. The space is "free" if there have been no events in the past seven minutes. Our extracted dataset contains the occupancy state ("free" or "occupied") of 50 sensors with the highest periodicity in [22], with a time resolution of 1 second.

6.3 Parameters

In this section we specify the choice of parameters during the experiments. We run our experiments on the ETH and MERL data sets for an evaluation interval of 3 months between April and June 2007 using a time window of 16 weeks (one semester) for the input time series. The time resolution used in these experiments is one hour and the maximum forecasting horizon is set to 168 hours or a week. For the Single-Period Model (SPM), we set the period length to one week, since we assume the periodic processes that dominate the time series of both datasets follow a weekly pattern.

For the Convolution Prediction Model (CPM) we need to define the period dominance threshold δ. As it is reasonable to assume that all relevant period lengths are multiples of one day in our datasets, the minimum period dominance for our data sets will be in the order of 0.7 because most rooms and office spaces are free outside the working hours (which accounts for a fraction of about 0.7 of 24 hours), which is a strong periodic pattern for any period length that is a multiple of one day. Hence, dominant periods can be expected to show a support greater than 70%. Initial experiments suggest that $\delta = 0.80$ prunes most of the irrelevant period lengths while keeping the important indicative periodic symbols in the set of mined periodicities $\mathcal{PS}'$.

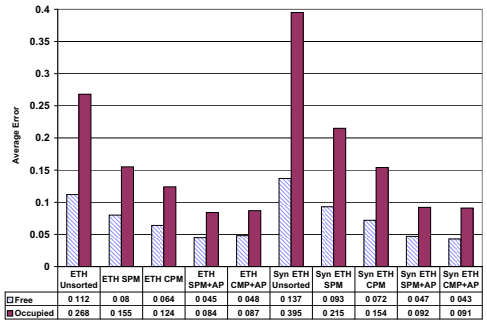

Figure 2: Average ranking error using different models on ETH data set and ETH synthetic data set.

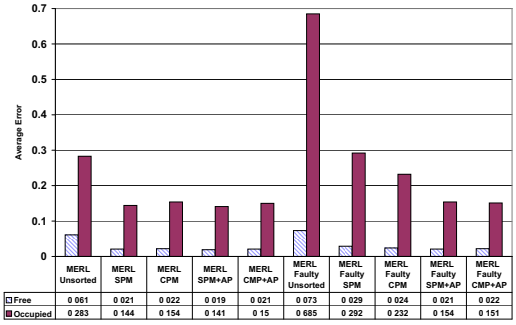

Figure 3: Average ranking error using different models on MERL data set, with and without the faulty sensor.

Further, CPM requires parameters for the minimum and maximum number of period instances ω_{min} and ω_{max} for computing the support of periodic symbols. Initial experiments on both data sets suggest that $\omega_{max} = 6$ gives the best results. As it is reasonable to assume that a day and a week are dominating period lengths in the two data sets, this maps to an effective time window of six days for a one day period, and six weeks for a one week period, which is reasonable given the nature of the data sets. Also through experiments we found $\omega_{min} = 3$ to give best results.

We also compared the performance of CPM with these parameters to the original version of the algorithm in [9] and found a reduction of the ranking error in the order of 25% for both data sets.

6.4 Results

Figure 2 and 3 show the average ranking error for queries posed for the two states "free" and "occupied", using different models for evaluation over ETH and MERL.

We show the results for Single-Period Model (SPM) and Convolution Prediction Model (CPM), both with and without the application of the adjustment process introduced in Sect. 5. As a baseline for comparison we include the unsorted case, where no sensor ranking is applied and the error metric is computed on a randomized list of sensors. In particular, we reshuffle the list of sensors after each query using the Fisher-Yates algorithm [15]. As we compute averages over many queries, this approach eliminates bias that may be introduced by a specific ordering of the sensors.

Recall from Sect. 3 that the ranking error indicates the fraction of the non-matching sensors that have to be checked by the search engine. For example, to find free rooms in the MERL dataset in the unsorted case, on average about 6% of all checked sensors actually do not match the query. When using the SPM model, only about 2% non-matching sensors have to be checked.

Note that the overhead for finding occupied sensors is generally larger than that for finding free sensors. The reason is that on average over all sensors and over the experiment duration, a sensor is much more likely to be free (e.g., due to nights and weekends) than occupied. Hence, on average it is easier to find a free room than to find an occupied room.

A further observation is that without the adjustment process, the CPM model performs better than the SPM model on the ETH data sets, while CPM and SPM show similar performance on the MERL data set. The reason for this is that sensors in the MERL data set monitor offices with a dominant period length of one week: with few exceptions, people work from Monday to Friday and stay home on the weekend. Hence, there is not much to be gained from using a prediction model such as CPM that can deal with multiple periods. In contrast, the ETH data has been obtained from seminar and meeting rooms. While most activities also follow a weekly pattern, there are certain meetings which take place once in two or four weeks, violating SPM's assumption of a single dominant period. Hence, CPM performs better than SPM on this data set.

To further investigate the potential of CPM, we have created a variant of the ETH data set called ETH synthetic, where one of the "sensors" is modified such that it has two equally dominant period lengths of two and three weeks, respectively. For this, we create two copies of the output of one of the sensors with the time axis scaled by a factor of two and tree, respectively. Then, we merge these two copies using the following rule: the output of the merged sensor at time t equals "occupied" if and only if at least one of the scaled sensors is "occupied" at t. Otherwise, the output is "free". As shown in Fig. 2, the improvement of CPM over SPM is larger for this synthetic data set when compared to the original ETH data set.

When applying the adjustment process, CPM and SPM achieve comparable results on both data sets. While for the MERL data set adjustment does not show a significant improvement, it is quite effective on the ETH data set. The reason for that and for the relatively greater improvement achieved by the adjustment process for SPM (compared to CPM) lies in the fact that the presence of period lengths different from one week leads to a poor accuracy of probability estimates computed by SPM. For example, if a room is occupied every other Monday, SPM will compute a probability of

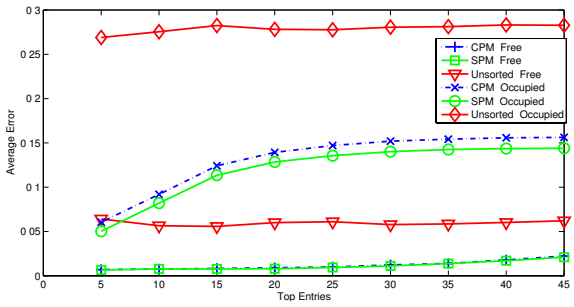

Figure 4: Comparison of the average error for the top entries of the rank list.

0.5 for the room being occupied on Monday, which is wrong for both odd Mondays (where it should be one) and even Mondays (where it should be zero). The adjustment process is able to correct the resulting persistent systematic error. However, recall from Sect. 5 that adjustment only works well when the query is executed rather frequently, which is the case in our experiments, but may not be the case in realistic settings.

For both the ETH data set and the MERL data set, there exists some form of homogeneity in the list of candidate sensors. Sensors in our two data sets follow similar periodic patterns with similar dominant period lengths of a week or multiples of it. For both of our two data sets the state during nights and weekends is usually "free" and activities occur during the working hours of the day. This similarity in the periodic patterns of the sensors reduces the average error for the unsorted case, since for some portion of the time there is no match for the queries or every sensor matches the query, and hence, the ranking error is equal to zero.

In the presented experiments, the subset of sensors that match the static part of the query, or the candidate list, is usually composed of sensors that monitor entities who share similar properties. Although it could be expected that this subset of sensors follow comparable periodic patterns, in the real world settings there may be inaccurate data gathered by some faulty sensors, or there are some entities whose behavior follow a completely different pattern. This introduces some heterogeneity in the underlying periodic processes of the candidate sensors and could increases the average ranking error in the unsorted case. During initial experiments with the MERL data set, we faced a special case where a sensor had stopped producing data half way through the evaluation interval, and constantly showed the state "occupied" afterwards. Figure 3 shows the results when this faulty sensor is included, displaying a larger average ranking error for the state "occupied" for the unsorted case in comparison to the experiment with the MERL data set, excluding this faulty sensor. We see that with the introduction of a sudden change of schedule, the CPM model shows a larger improvement in the ranking error over the SPM. We believe that the time window selection with ω_{max} in the CPM model allows it to adapt faster to this change of schedule.

So far, we have considered the case where we need to find all sensors that match a given query. However, as noted in Sect. 3, often it is sufficient to find a small subset of sensors matching the query. This is where sensor ranking unfolds its full potential. Fig. 4 shows average ranking errors using the top-m metric introduced in Sect. 3, where we only consider the m top-ranked sensors. The diagram shows the ranking error for the MERL data set, the parameter on the x-axis is the number m. One can see that the ranking error generally decreases if we consider smaller m, which means that the top-ranking entries have a relatively high probability of being matches. In the extreme case of $m = 5$ both CPM and SPM achieve an improvement of a factor of 10 (5) over the unsorted case for the "free" ("occupied") state.

Finally, Fig. 5 shows the average ranking error as a function of the forecasting horizon for the MERL data set with the inclusion of the faulty sensor. As the maximum forecasting horizon in our experiments is one week (i.e., 168 hours), we can see the daily activity patterns. One can clearly identify the nights when the error drops significantly. We can see that the heterogeneity introduced by the inclusion of the faulty sensor increases the overall ranking error for the unsorted case, especially during the night and weekends. The prediction models are able to reduce the effect of this faulty sensor, however, the error does not drop to zero because it takes some time for the prediction models to adapt to the change of schedule introduced by the faulty sensor. We can see that CPM performs better than SPM in this regard, mirroring the results from Fig. 3. The application of the adjustment process is able to further reduce the persistent error that is present during the nights and the weekends.

Summarizing, we found that sensor ranking can provide an improvement in the order of one magnitude over the baseline case when searching for a small subset of sensors that match a query. The MERL data set has a dominant period length of one week, such that SPM's assumption of a single dominant period is met, resulting in similar performance of SPM and CPM. The ETH data set includes a more diverse set of period lengths, such that CPM can gain an improvement over SPM. By further increasing the diversity of period lengths by including a synthetic sensor in the ETH data set, the difference between SPM and CPM is further pronounced. Experiments with inclusion of a faulty sensor to the MERL data set suggests that reducing the homogeneity of sensors increases the average error in the unsorted case. If queries are executed frequently, the adjustment process can result in substantial improvements, especially in the presence of persistent inaccurate predictions for some sensors.

7. RELATED WORK

Some work exists regarding web search engines that can search for dynamic Web content. In [11], a system is pro-

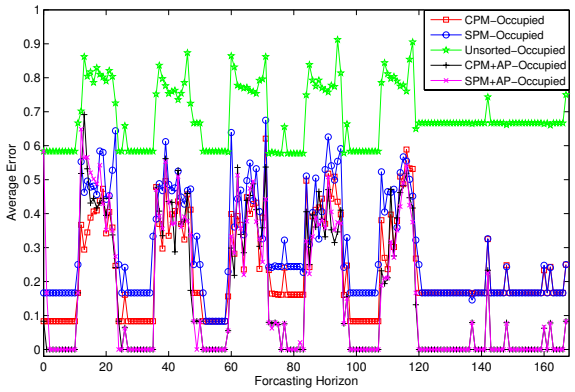

Figure 5: The average error over a maximum forecasting horizon of a week.

posed that enables real-time search of web pages by using a crawler for each request. For this, besides the actual search term, starting points of the search in form of a list of URLs and a time limit have to be specified in a SQL-like search language. A very similar approach is taken in [21]. However, both approaches are not feasible to be applied to a Web of Things due to its expected size.

Further web search engines that support dynamic Web content include Google News [1] and Technorati [2]. While the search space of the former is rather limited, as only professional news sites are considered, Technorati claims that they are currently tracking 112.8 million blogs. However, Technorati seems to exploit certain hints to realize a near real-time search, for example, by offering a web service (a *RPC-Ping*) which is utilized by bloggers to inform the search engine about new posts [3]. However, due to the expected size of a Web of Things and frequent changes of sensor output, we believe this approach is not applicable to the WoT.

Middleware such as GSN [4] or SenseWeb [14] offers unified interfaces to access heterogeneous sensors and sensor networks and to execute queries on the sensor data streams. However, these systems do not address content-based sensor search. Instead, sensors can only be selected based on their static meta information (i.e., type or location of the sensor).

In [18, 20] a search engine for the physical world has been proposed which allows to index static information stored in sensor nodes and to search for sensor nodes that store a particular static information. However, this system does not support search for highly dynamic sensor output which is the focus of our work.

Several projects investigate the output of people-centric sensors (e.g., [17, 8]), in particular they study the periodic nature and predictability of the data. However, they make no effort to provide content-based sensor search.

Prediction models have been applied in the context of sensor networks to reduce the amount of data that has to be transmitted over the wireless network (e.g., [7, 19]). However, the prediction models used there differ substantially from our work, as they focus on short-term prediction and do not support categorical time series. Also, these models produce a predicted sensor value as output, while our models produce probability that the sensor outputs a given value.

A survey of prediction methods for categorial time series in [16] concludes that most existing methods are not suitable for large scale and online applications. The convolution pattern mining algorithm introduced by [9] – the foundation of our work – is considered to be a promising approach.

8. CONCLUSIONS

We have introduced the *sensor ranking* primitive for efficient content-based sensor search as an important building block of a search engine for the Web of Things, where real-time state information of real-world entities can be published and searched. The key idea of sensor ranking is to exploit the periodic nature of people-centric sensors by using appropriate prediction models. Using these prediction models, we can rank sensors according to their probability of matching a content-based sensor search. Using two real-world data sets, we could show that sensor ranking can significantly improve the performance of a search engine compared to a baseline method. Future work includes improvements of the prediction models (e.g., automatic estimation of model parameters and model selection) and incorporating sensor ranking into a search engine for the WoT.

9. REFERENCES

[1] Google News Search. `http://news.google.de`.
[2] Technorati. `http://www.technorati.com`.
[3] Technorati Ping Configurations. http://technorati.com/developers/ping/.
[4] Karl Aberer, Manfred Hauswirth, and Ali Salehi. Infrastructure for data processing in large-scale interconnected sensor networks. In *MDM*, 2007.
[5] Karl Abrahamson. Generalized string matching. *SIAM J. Comput.*, 16(6):1039–1051, 1987.
[6] Christos Berberidis, Walid G. Aref, Mikhail J. Atallah, Ioannis P. Vlahavas, and Ahmed K. Elmagarmid. Multiple and partial periodicity mining in time series databases. In *ECAI*, pages 370–374, 2002.
[7] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Modeldriven data acquisition in sensor networks. In *VLDB 2004*.
[8] Nathan Eagle and Alex (Sandy) Pentland. Reality mining: Sensing complex social systems. *Personal Ubiquitous Comput.*, 10(4):255–268, 2006.
[9] Mohamed G. Elfeky, Walid G. Aref, and Ahmed K. Elmagarmid. Using convolution to mine obscure periodic patterns in one pass. In *EDBT*, pages 605–620, 2004.
[10] ETH CS Dpt. Online Room Reservation System. `http://www.bookings.inf.ethz.ch`.
[11] Augustine Chidi Ikeji and Farshad Fotouhi. An adaptive real-time web search engine. In *WIDM '99*, pages 12–16. ACM, 1999.
[12] Piotr Indyk, Nick Koudas, and S. Muthukrishnan. Identifying representative trends in massive time series data sets using sketches. In *VLDB '00*, pages 363–372. Morgan Kaufmann Publishers Inc., 2000.
[13] Bernard J. Jansen, Amanda Spink, Judy Bateman, and Tefko Saracevic. Real life information retrieval: a study of user queries on the web. *SIGIR Forum*, 32(1):5–17, 1998.
[14] Aman Kansal, Suman Nath, Jie Liu, and Feng Zhao. Senseweb: An infrastructure for shared sensing. *IEEE MultiMedia*, 14(4):8–13, 2007.
[15] D. E. Knuth. *The art of computer programming, v. 2 (3rd ed.): seminumerical algorithms*. 1997.
[16] R. Mayrhofer. *An Architecture for Context Prediction*. PhD thesis, Johannes Kepler University of Linz, Austria, October 2004.
[17] Jonathan Reades, Francesco Calabrese, Andres Sevtsuk, and Carlo Ratti. Cellular Census: Explorations in Urban Data Collection. *IEEE Pervasive Computing*, 6(3):30–38, 2007.
[18] C. C. Tan, B. Sheng, H. Wanh, and Q. Li. Microsearch: When search engines meet small devices. In *Pervasive 2008*.
[19] D. Tulone and S. Madden. Paq: Time series forecasting for approximate query answering in sensor networks. In *EWSN 2006*.
[20] H. Wang, C. C. Tan, and Q. Li. Snoogle: A search engine for the physical world. In *Infocom 2008*.
[21] Burr S. Watters. Development and performance evaluation of a real time web search engine. Master's thesis, University of North Florida, December 2004.
[22] Christopher R. Wren, Yuri A. Ivanov, Darren Leigh, and Jonathan Westhues. The merl motion detector dataset. In *MD '07: Proceedings of the 2007 workshop on Massive datasets*, pages 10–14. ACM, 2007.