

# Publish/Subscribe Paradigm for Time Series Data: Evaluation of Two Spatial Search Algorithms

Kaloian Manassiev  
Department of Computer Science  
University of Toronto  
kaloianm@cs.toronto.edu

Hans-Arno Jacobsen  
Department of Computer Science and  
Department of Electrical and Computer Engineering  
University of Toronto  
jacobsen@eecg.toronto.edu

## Abstract

In the publish/subscribe paradigm, data providers disseminate publications to all consumers that have registered subscriptions. Most of the research in publish/subscribe systems is focused on semi-structured data where publications are represented as a set of  $\langle attribute, value \rangle$  pairs and subscriptions express constraints over attribute values. In this paper we introduce a publish/subscribe paradigm for time series data. Publications and subscriptions in this model correspond to numerical sequences of fixed length and the semantics of matching is based on the notion of *similarity* between such sequences. Since time series can be modeled as higher dimensional points we evaluate the applicability of an existing spatial data indexing approach for the support of matching in such type of system. We provide a framework for developing time series publish/subscribe middleware which domain experts could easily extend to fit a particular application.

## 1 Introduction

The publish/subscribe paradigm defines a selective data dissemination model which consists of publishers, subscribers and a central broker or a distributed network of brokers. Data is produced by publishers and, according to certain dissemination rules, eventually gets delivered to interested subscribers. Subscribers express their interest using *subscriptions*, which impose restrictions on the data content. The role of the broker network is to find all subscriptions which match an incoming publication and deliver it only to subscribers with matching subscriptions.

---

*Paper submitted to 2nd Workshop on Spatio-Temporal Database Management, May 2004. Kaloian Manassiev, Hans-Arno Jacobsen, Copyright 2004.*

The type of publication data that has gained most research attention presently corresponds to a set of  $\langle attribute, value \rangle$  pairs. Each attribute in the set has unique name and fixed type which determines the universe of its values. Subscriptions consist of predicates, usually a conjunction of boolean operators expressing some restriction on the attribute values. For example, the publication  $\{(type, bachelor), (location, downtown), (price, \$800)\}$  satisfies the subscription  $\{type = any \wedge location = downtown \wedge price \leq \$1000\}$ . Publish/subscribe systems have been developed for selective dissemination of such type of data in both centralized [15, 4] and distributed [3] aspect.

In this paper we introduce a publish/subscribe paradigm for time series data. Subscriptions and publications in this model are numerical sequences of fixed length. We base the semantics of matching on the notion of similarity between the time series of the subscription and those of the publication. For the matching part we use approaches developed for queries in sequence and geo-spatial databases [9, 2] and conduct experiments to evaluate their applicability for solving the time series publish/subscribe problem.

Various real-world data is represented as time series. Stock fluctuations, seismic activities, sound and images are just a few examples. Below we present some scenarios where a time series publish/subscribe middleware could find good utilization.

**Stock fluctuations** are usually modeled as time series. Data in series represents the change of the market value of a particular stock, recorded at regular time intervals. If we plot this time series in the two dimensional plane, we will obtain the *shape* of that stock. The study of shapes is very important in financial market analysis [7, 11].

For the above type of data a subscription could be a sample time series, which reflects the sought trends shape and some tolerance threshold which indicates the desired similarity level. Marketing

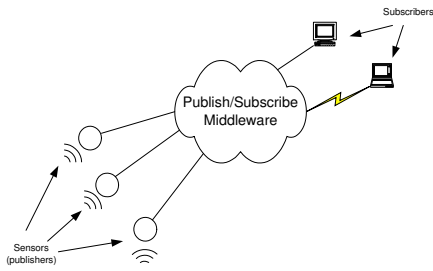


Figure 1: Acoustic sensor network with publish/subscribe middleware.

researcher could pose such a subscription to a stock brokerage company. Its semantics will state *send an event any time you deal with stocks, which have history trends similar to that of the sample.*

**Sound signals** represent the air pressure generated by a sound source as a function of time. The variation of the value of a signal with time gives its waveform. Different sounds in real-life have different waveforms. For example, the waveform generated by a passing truck is different from that of a passing car. Time series publish/subscribe systems could find good application in a network of sound sensors.

Surveillance personnel looking for a particular type of object passing through a monitored area could pose a subscription stating *inform me when any sensor reports a waveform that is similar to a given one.* The sample waveform could be that of a truck, car or an animal. Figure 1 depicts a publish/subscribe system with a number of acoustic sensors. Sensors attached to the broker act as publishers, while the two PC nodes act as subscribers.

In the process of our work we built a framework for development of time series matching engines, which we used for the evaluation of the proposed algorithms. This framework is generic and can easily be extended to support applications from different domains.

We continue this paper with a background on time series similarity matching, feature extraction and indexing. Section 3 defines the semantics of subscriptions and publications and describes the matching engine on a conceptual level. It also outlines the architecture of our time series publish/subscribe system. Section 4 describes the matching engine, which we developed in three variants, based on  $R^*$ -tree [2], interval tree [6] and simple linear scan. We evaluate and discuss the performance of these algorithms in Section 5. Section 6 concludes the paper and gives some future work directions.

## 2 Background

*Time series* are real-valued sequences representing measurements taken at regular time intervals. Such data is inherently inexact due to the fixed precision supported by computers and the general imprecision of sensing devices. For that reason, most of the operations for time series comparison are *approximate*. The operation we are going to discuss in this paper is the *shape similarity matching* [9].

At this early stage of our work will consider only *whole matching* whereby the sequences to be matched each have the same length  $n$ . For brevity, we will adopt the term  $n$ -time series ( $n$ -sequence) to refer to the vector  $X = \langle x_1, x_2, \dots, x_n \rangle$  of length  $n$ . Also, whenever we use the terms *distance* or *norm*, unless explicitly stated, they will mean *Euclidean distance* and  $L_2$  norm in the  $n$ -dimensional Euclidean space.

### 2.1 Distance and Similarity

Let  $X$  and  $Y$  be two  $n$ -time series and let  $D$  be a distance metric. It is said [1, 9] that  $X$  and  $Y$  are similar with respect to  $D$  and tolerance  $\epsilon$  if:

$$D(X, Y) \leq \epsilon, \text{ for } \epsilon > 0. \quad (1)$$

The distance metric  $D$  is usually chosen with regard to the application domain. In this paper we will use the  $L_2$  norm as it is non-negative, obeys the triangle inequality and is efficiently computable. The  $L_2$  norm defines the Euclidean distance between two  $n$ -time series:

$$D(X, Y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2} \quad (2)$$

### 2.2 Similarity Semantics

Human perceived *similarity* is subjective and varies with the type of data and the application domain. For example, for time series representing image colour histograms, the definition of similarity given in Eq. 1 and 2 is sufficient. However, for other type of data, such as stock prices, it is more natural to seek similarity in terms of shapes. Given two stocks, the price of the first one might be twice the price of the other. Such series are not similar in Euclidean sense, but they have similar *shapes*.

#### 2.2.1 Normal Forms and Shape Similarity

For a given  $n$ -time series  $X$ , the transform:

$$T_{a,b}(X) = aX + b, \text{ where } a, b \in R, a \neq 0 \quad (3)$$

preserves the shape of  $X$  [9, 17]. In the above equation,  $a$  is the *scale* and  $b$  is the *shift* of the transform.

**Definition 1** Let  $\alpha(X)$  and  $\sigma(X)$  be the mean and the standard deviation of  $X$  respectively. An  $n$ -sequence  $X$  is said to be in normal form if  $\alpha(X) = 0$  and  $\sigma(X) = 1$ . Thus, given  $X$ , we can obtain its normal form  $N(X)$  with the following transformation [9]:

$$N(X) = T_{1/\sigma, -\alpha/\sigma}(X). \quad (4)$$

Thus, when  $X$  and  $Y$  are two  $n$ -time series and  $D$  is the  $L_2$  norm, the *shape similarity* between  $X$  and  $Y$  is defined as the Euclidean distance between their *normal forms* [9]:

$$D_S(X, Y) = D(N(X), N(Y)). \quad (5)$$

Using Eq. 5, it can be said that  $X$  and  $Y$  are *shape similar* with tolerance  $\epsilon$  if:

$$D_S(X, Y) \leq \epsilon, \text{ for } \epsilon > 0. \quad (6)$$

As mentioned in [9],  $\epsilon$  should be interpreted as the maximum level of dissimilarity. For this paper we chose to use the expression “*shape similar* with tolerance  $\epsilon$ ”.

With the definitions above we can treat any  $n$ -time series as a point in the  $n$ -dimensional Euclidean space. For clarity, further in this paper we will use the terms  $n$ -time series and  $n$ -dimensional point interchangeably.

### 2.3 Dimension Reduction

Computing the Euclidean distance between two  $n$ -sequences takes  $O(n)$  operations on average. When time series are longer and the volume of data is high we cannot afford to compute the similarity distance between a sample sequence and all other sequences. As the *dimensionality curse* cannot be avoided, some technique should be used to quickly reduce the number of  $n$ -sequences that need to be examined.

The method most commonly used for *fast rejection* in sequence databases [1, 9, 19] employs a *feature extraction function*  $F : R^n \rightarrow R^k$ , which projects each  $n$ -dimensional point onto a much lower dimensional space.  $F$  is chosen such that whenever the projections of two time series are not similar, the original series are not similar as well. For brevity, we will refer to the  $n$ -dimensional space of the time series as the *series space*, to the lower dimensional space as the *feature space* and to the projection of an  $n$ -series onto the *feature space* as the *feature vector*.

We will use the first few coefficients of the Fast Fourier Transform (FFT) [16] of  $X$  as a *feature extraction function*:

$$F(X) = [FFT_1(N(X)), \dots, FFT_k(N(X))]. \quad (7)$$

Here,  $FFT_i(N(X))$  denotes the  $i^{th}$  coefficient of the Fast Fourier Transform of the *normal form* of  $X$ .

FFT is widely used and has been shown to provide very good accuracy for  $k = 2$  to 5 [1, 19]. It should be

noted that Fourier coefficients are complex numbers, so the resultant *feature space* will be  $2 * k$ -dimensional. Also, the first Fourier coefficient of any normalized sequence is zero [16] thus we will not include  $FFT_0$  in the *feature vector*.

Following the model from [9] we will define the *feature space* similarity function as

$$D_F(X, Y) = D_S(F(X), F(Y)). \quad (8)$$

Using Parseval’s Theorem [16] and the monotonic property of the Euclidean distance it can be shown that this indexing approach is *correct*:

$$D(X, Y) \leq \epsilon \Rightarrow D_F(X, Y) \leq \epsilon. \quad (9)$$

To improve performance, Rafiei *et al.* [19] exploit the fact that Fourier coefficients of any real-valued signal of length  $n$  satisfy  $X_i = X_{\frac{n}{2}+i}^*$  ( $X^*$  is the complex conjugate of  $X$ ) and obtain:

$$D(X, Y) \leq \epsilon \Rightarrow D_F(X, Y) \leq \epsilon/\sqrt{2}. \quad (10)$$

## 3 Publish/Subscribe Model

In this section we describe the subscription and publication structures in our system and define the semantics of matching.

**Subscriptions** are triples in the form  $S = \langle I, X, \epsilon \rangle$ , where  $X = (x_1, x_2, \dots, x_n)$  is an  $n$ -time series,  $\epsilon$  is a tolerance and  $I$  is a generic field used to uniquely identify the subscriber who posted  $S$ .

**Publications** are  $n$ -time series  $P = (v_1, v_2, \dots, v_n)$ .

### 3.1 Matching Problem

**Definition 2** We will say that publication  $P$  matches a subscription  $S = \langle I, X, \epsilon \rangle$  if  $X$  and  $P$  are shape similar with tolerance  $\epsilon$ , e.g.,

$$D_S(X, P) \leq \epsilon. \quad (11)$$

Let  $P$  be a publication and  $\mathcal{S} = \langle S_1, S_2, \dots, S_m \rangle$  be a set of subscriptions. The publish/subscribe problem for time series data can be stated as follows: Given  $P$  and  $\mathcal{S}$ , find the subset  $M(P) \subseteq \mathcal{S}$  of subscriptions, which match  $P$  in the sense of Def. 2.

From Eq. 10 we have that

$$S \in M(P) \Rightarrow D_F(X, P) \leq \epsilon/\sqrt{2}. \quad (12)$$

In order to speed the search, we will index each subscription and each incoming publication with their *feature vectors*:

$$S_f = \langle X_f, \epsilon/\sqrt{2}, S \rangle, \quad X_f = F(N(X)). \quad (13)$$

$$P_f = F(N(P)). \quad (14)$$

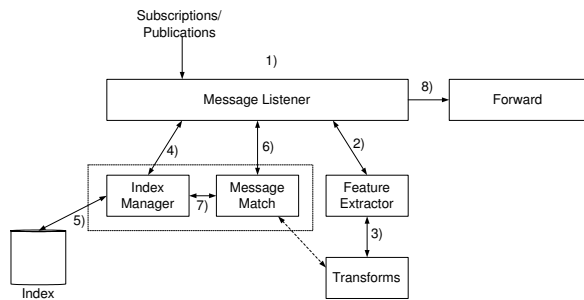


Figure 2: Broker node organization.

That way, given the set  $\mathcal{S}$  we will be able to quickly find the subset  $\mathcal{S}' \subseteq \mathcal{S}$  of subscriptions, which certainly do **not** match  $P$ . The remaining subscriptions form the *candidate list*  $\mathcal{S}_C = \mathcal{S} - \mathcal{S}'$ . Unfortunately, all the subscriptions from  $\mathcal{S}_C$  will need to be matched against  $P$  in the  $n$ -dimensional *series space*.

### 3.2 Broker Nodes

Broker nodes in a publish/subscribe system serve as a connection between publishers and subscribers. Each broker has a number of input interfaces through which publications and subscriptions are posted, and a number of output interfaces through which matched publications are sent to the interested subscribers. In a distributed system brokers may also connect to other brokers in order to relay data.

Figure 2 depicts the main components of a broker node in the time series publish/subscribe system. The operation flow consists of the following steps:

1. Broker receives an incoming subscription or publication on some of its input interfaces.
2. It is sent to the feature extractor component to obtain its feature vector (cf. Section 2.3).
3. The feature extractor uses algorithms exposed by the *Transforms Library* to extract the feature vector. In our case, this is the Fast Fourier Transform, but at domain expert's will it could be changed to any other transform which suits the application domain.
4. If a **subscription** has been received, its feature vector is passed to the index manager.
5. The index manager updates the Index Structure.
6. If a **publication** has been received, it is passed to the matching engine, which retrieves the set of subscribers to which it should be forwarded.
7. Matching engine uses the index to achieve the above task.
8. Finally, the publication is forwarded to each of the matched subscribers through the respective output interfaces.

## 4 Matching

Even if we are working in the  $2 * k$ -dimensional *feature space*, it would still be good if there is some technique that could trim the search space, so that just a small

fraction of the *feature representations* of subscriptions need to be tested.

Using Eq. 13 and 14, each subscription's *feature representation*  $S_f$  can be modeled as a  $2 * k$ -dimensional hypersphere  $\Sigma$  of radius  $\epsilon/\sqrt{2}$ .  $P_f$  matches  $S_f$  in *feature space* when  $P_f$  is in  $\Sigma$  (or on its boundary). With this representation, we can optimize the discovery of the *candidate list*  $\mathcal{S}_C$  by modeling the search as a high-dimensional *point query* [22, 21].

The *point query* asks for *all objects which overlap a given reference point* and there is a variety of algorithms which are related to answering it efficiently. Below we give a brief overview of the algorithms, which we looked at as potentially suitable for solving this problem. Exhaustive survey is available in [8].

*Point Access Methods* or *PAMs* have been designed as an indexing tool for databases that consist solely of point data. Representative of these methods are the *grid file* [18] and the *k-d-B tree* [20]. All of these methods exploit the fact that points do not have spatial extent to achieve an optimal search space partitioning. However, since subscriptions are spatial objects, we considered that *PAMs* are impractical to use as a subscription index.

*Spatial Access Methods* or *SAMs* have been developed exclusively for the management of objects having spatial extent, such as lines, polygons or polyhedra. Some of these methods do partitioning of the object space into regions in order to prune the search, while other use a balanced tree approach whereby leaf nodes contain the actual objects and intermediate nodes contain a bounding approximation of their children. The most widely used tree-based structures include *R-Tree* [10], *R\*-Tree* [2], *SS-Tree* [14] and *SR-Tree* [13].

For the purposes of our experiments we decided to use the *R\*-Tree* as one of our subscription indexing structures because it is simple and its code is readily available. Also, the *R\*-Tree* has shown little performance degradation with higher-dimensional data and has successfully been used in sequence database applications [1, 19].

### 4.1 Subscription Indexing Techniques

The following two subsections describe the indexing techniques we used for optimizing the discovery of the *candidate list*  $\mathcal{S}_C$ .

#### 4.1.1 R\*-Tree Index

The *R\*-tree* [2] is a height balanced tree, similar to a B-tree [6]. Based on the *R-Tree* [10], it indexes multi-dimensional data objects by approximating them with their iso-oriented minimal bounding boxes (*MBBs*). Leaf nodes contain pointers to the actual data objects, while internal nodes consist of the minimal bounding boxes of their children.

In our system, the data objects were the subscription *feature representations*  $S_f$ . Since *R\*-Tree* indexes

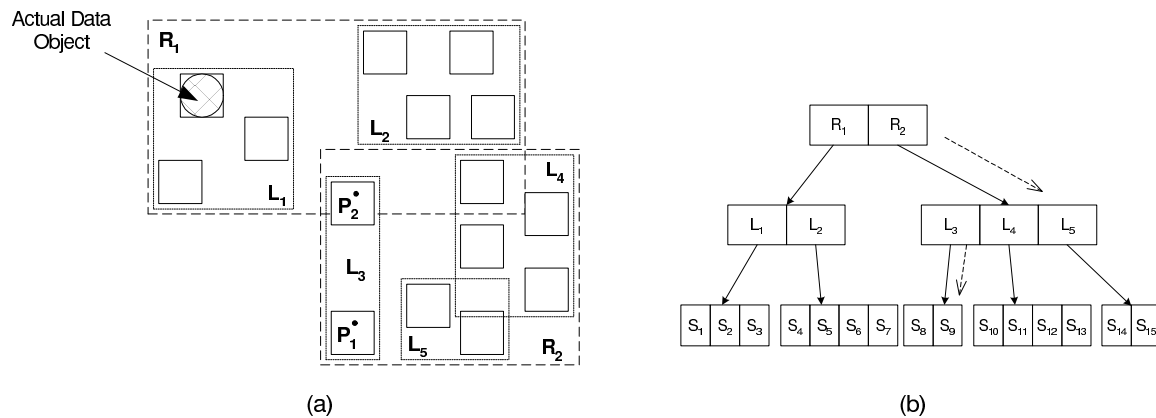


Figure 3: R-tree organization.

rectangles, we approximated all subscription spheres with their *MBBs* prior to inserting them in the index. Figure 3a shows the structure of an *R\*-tree*, which contains 15 planar spheres (for clarity, only one sphere is drawn). On the diagram,  $L_1, L_2, L_3$  and  $L_4$  are the leaf nodes with dotted lines depicting their minimal bounding rectangles. Likewise,  $R_1$  and  $R_2$  are internal nodes with the dashed lines depicting their minimal bounding rectangles.

In order to obtain the *candidate list*  $S_C$  of subscriptions matching a given publication *feature vector*  $P_f$ , we impose a *point query* to the *R\*-tree* passing  $P_f$  as the point. The result of the query is the set of sphere objects, which contained the point  $P_f$ .

Following the *R\*-Tree* search algorithm, the match starts at the root node. At each level it chooses only child nodes whose bounding rectangles contain the query point  $P_f$ . When a *leaf* node is reached, all subscriptions it contains are examined by scanning them linearly and only those that are similar to  $P_f$  are inserted in the *candidate list*. For example, the dashed lines in Figure 3b show the search path for subscriptions that match point  $P_1$ .

It follows from the above that higher overlap between rectangles in internal nodes would deteriorate the search time. For example, the search for subscriptions that match  $P_2$  in Figure 3b will examine all leaf nodes of both  $R_1$  and  $R_2$  although only the leaf  $L_3$  of  $R_2$  contains the reference point.

Due to space considerations we will not give further details of the algorithms that operate on the *R-Tree* and the *R\*-Tree*. They are available in the original papers [10, 2].

#### 4.1.2 Interval Tree Index

As an alternative to the technique described in the previous section, we decided to use only two of the point coordinates to trim the search for the *candidate list*, and compare the performance of this approach to the *R\*-Tree*.

For any  $k$ -dimensional point  $P = \langle x_1, \dots, x_k \rangle$  con-

tained within the *minimal bounding box* of a sphere  $\Sigma = (\langle y_1, \dots, y_k \rangle, \epsilon)$  of radius  $\epsilon$ , we have:

$$x_i \in [y_i - \epsilon, y_i + \epsilon], i = 1 \dots d. \quad (15)$$

Observing this, we can use two *interval trees* to index the set of subscription *feature vectors* based on their first Fourier coefficient (recall that Fourier coefficients are complex numbers).

The *interval tree* [6] is a red-black tree, which maintains a dynamic set of one-dimensional intervals. Each interval is assigned a place in the red-black tree using its start or end point as a key. The height of an  $n$ -node red-black tree is  $O(\lg(n))$ , so given a point  $p$ , the *search* for a single interval that contains  $p$  takes  $O(\lg(n))$  time. Figure 4 depicts an *interval tree*.

We implemented the interval tree search algorithm so that it returns *all* intervals, which contain a given query point. Thus, for a given publication *feature vector*  $P_f$  the so built index will immediately reject any subscriptions, which “*span too far*” in their first two coordinates. The rest of the subscriptions still have to be tested using linear scanning.

## 5 Evaluation

All the techniques described in the preceding sections were implemented in C++. We used our own implementation of the Interval Tree. For the experiments with the *R\*-Tree* we used sources obtained from [12]<sup>1</sup>. The *R\*-tree* was implemented with secondary storage in mind and used wrapper classes and quite long function call chain, which caused noticeable delay in the processing speed when used in-memory. For that reason, we also give our results in terms of the number of series actually tested, which is an indicator of the pruning abilities of the proposed indexing algorithms. The split threshold of the *R\*-Tree* was 24 child nodes and the reinsert factor was 30%.

<sup>1</sup>We are very thankful to Norio Katayama *et al.* for publishing their code on the web.

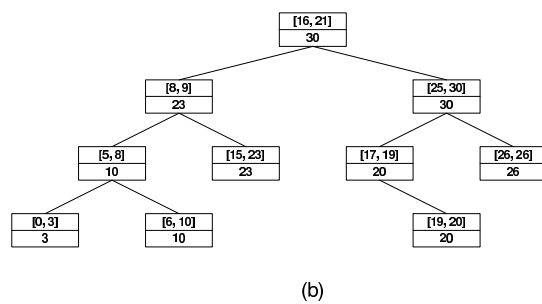
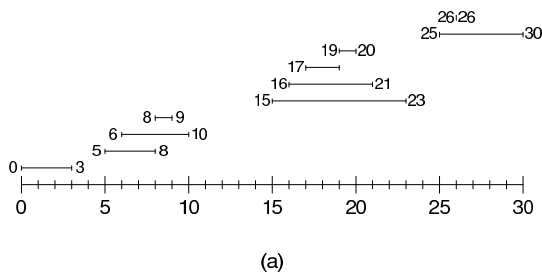


Figure 4: Interval tree.

Table 1: Workload parameters for series generation.

Parameter	Value Range	Description
N	1k, 2k, 5k, 10k, 50k, 100k	# of subscriptions
L	2040	# of samples in series
k	3, 4, 5, 6	# of FFT coefficients
c	(0.01, 1.0)	Tolerance scaling

### 5.1 Experimental Setup

We ran all our experiments on a Pentium 4 2.8GHz processor with 1.5GB of memory, running RedHat Linux 8.0. All our tests fitted entirely in main memory and the maximum memory usage we observed was 40% when *R\*-Tree* with 100000 subscriptions was used.

Table 1 lists the parameters we controlled and their respective value ranges.

*Random walks* have successfully been used to model stock movements and exchange rates [5] and have been used as test data in [1]. For that reason, as a test workload, we generated synthetic *random walks* data.

First, we generated  $N$  sequences consisting of  $L$  samples. Each sequence  $X = [x_i]$  was a random walk  $x_{i+1} = x_i + w_i$ , where  $w_i, i = 0, 1, \dots, L - 1$  were independent random variables, uniformly distributed in the range  $(-500, 500)$ .

For each sequence  $X$  we generated a random tolerance  $\epsilon$ , uniformly distributed in the interval  $(0, c \times \sqrt{1000 \times L})$ . Taking all pairs  $\langle X, \epsilon \rangle$  we constructed the set  $\mathcal{S}$  of subscriptions and indexed it using the methods described in the previous section.

For publication data, we generated additional  $\frac{N}{10}$  random walks of length  $L$  and for each of them calculated the *candidate list*  $S_C$ .

We ran each experiment several times, varying the parameter  $c$ . This yielded different average subscription radius at each run and thus affected the overlap between subscription spheres and increased the fraction of subscriptions that matched each publication.

For each of the runs we measured the size of  $S_C$ , the average subscription time, the average matching time and the number of subscriptions which had to be tested linearly in the search for the *candidate list*  $S_C$ .

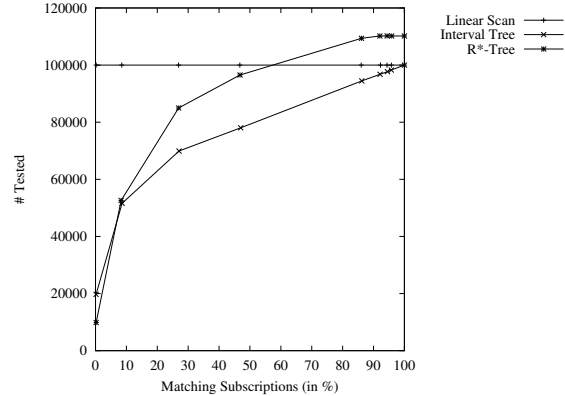


Figure 5: # Tested vs. Size of  $S_C$ ,  $N=100k$ ,  $k=5$ .

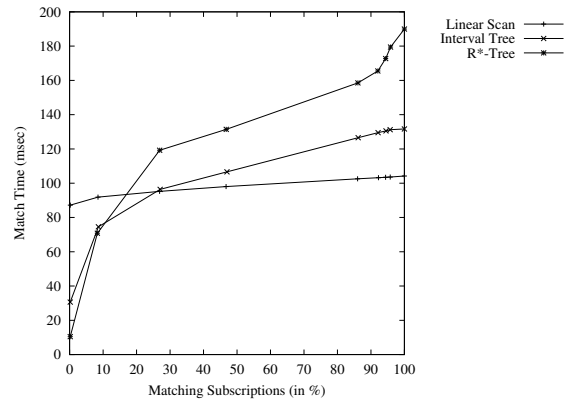


Figure 6: Match Time vs. Size of  $S_C$ ,  $N=100k$ ,  $k=5$ .

We present our findings in the following sections.

### 5.2 Pruning Capabilities and Matching Time

Figure 5 shows the average number of subscriptions that were actually tested at each run versus the size of  $S_C$ . The size of the subscription population was  $N = 100,000$  and the *feature vector* had 5 FFT coefficients ( $k = 5$ ). The linear scan case, where all series are tested is given just for reference. The size of  $S_C$  is given in percent of the whole subscription population. This can be interpreted as the “*fraction of useful tests*”.

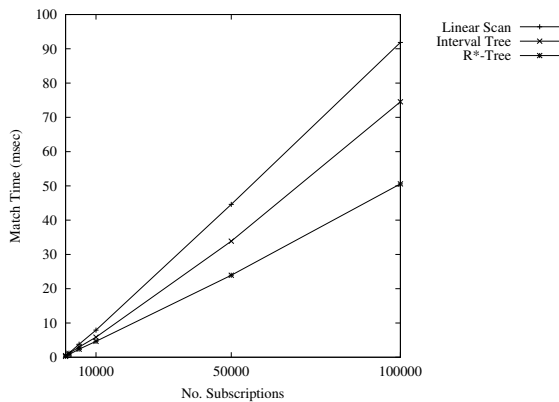


Figure 7: Match Time vs. # Subscriptions,  $N=100k$ ,  $k=5$ .

We considered that the test whether a point lies within a rectangle takes approximately the same number of operations as computing the Euclidean distance. For that reason, in  $R^*$ -Tree tests we counted each rectangle test as subscription test. This caused at some point the number of tested entries to exceed the total number of subscriptions.

From Figure 5 it can be seen that after the fraction of matching subscriptions exceeds 8%, the total number of tested nodes in the  $R^*$ -Tree starts to perform worse than the Interval Tree based algorithm in terms of search space pruning. This implies increased overlap between nodes at the same level of the  $R^*$ -Tree, which causes a large number of fruitless checks.

Figure 6 displays the average matching time against the size of  $S_C$  for the same parameters. The linear scan trace is not precisely horizontal because we used the best possible linear scan algorithm, whereby the computation stops as soon as the accumulated sum of squared differences exceeds the squared radius. In our tests, the  $R^*$ -Tree algorithm starts to perform much worse than both the linear scan and the Interval Tree, in terms of time, when the fraction of matching subscriptions reaches somewhere between 10 and 25 percent. We attribute this to the implementation factors described above and expect an in-memory version of this algorithm, which better accounts for locality of accesses to dominate in performance for at least 40% matching subscriptions.

Figure 7 displays the behaviour of the matching time as we increased the number of subscriptions in the system. For each data point displayed, approximately 8% of the subscriptions match.

### 5.3 Subscription Time

Finally, Figure 8 shows the variation of the subscribe time for different number of subscriptions in the system, *averaged* over the entire run. From this chart it can be inferred that the *average* subscribe time for both tree-based algorithms is not significantly affected

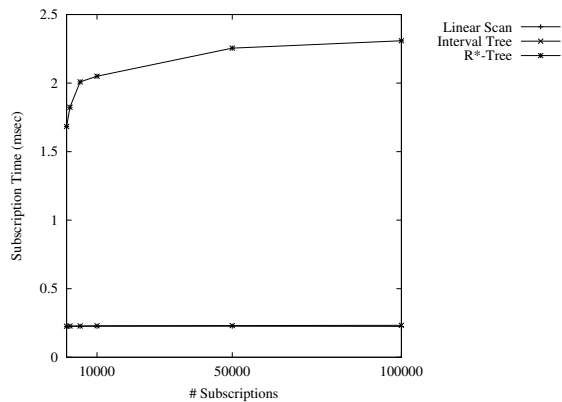


Figure 8: Subscribe Time vs. # Subscriptions,  $k=5$ .

by the size of the subscription population. For the  $R^*$ -Tree this is due to its “greedy” nature which, during object insertion, chooses a single subtree at each level, such that subtree’s bounding rectangle expansion will introduce the smallest overlap between neighbour nodes found at the same level. For the *interval tree* this is because of its  $O(\lg(n))$  insertion complexity.

## 6 Conclusion and Future Work

In this preliminary work we proposed a publish/subscribe model that operates on time series data and the semantics of matching is based on the notion of *similarity* between whole series. We used the *dimension reduction* approach in combination with *spatial indexing* to optimize the matching process and evaluated the performance of this technique using random walks data.

In the process of our work, we built a generic framework for developing time series publish/subscribe systems. It is easily extensible and allows expert in a given area to modify the components it needs to adjust it for a particular application. For example, changing the type of data to images will require changes to the feature extractor without need to modify the matching engine. In case certain knowledge about the type of data and respectively about the behaviour of feature vectors permits optimizing the matching process (for example, some coordinates are more important than other), only the matching engine needs to be changed.

Our experimental results showed that the time series matching process works better when there is a low number of matching entities. This is a result of the complex structure of spatial data and the inability of *spatial indexing structures* to take into account subscription overlaps. This is in contrast to semi-structured data based publish/subscribe systems, which model the overlap between subscriptions with a common path within a finite state automaton or a common predicate, and thus manage to match/discard significant number of subscriptions at a single pass.

As an immediate future work we plan to extend this scheme and evaluate its performance with real data, such as sounds or images from everyday life. As a result of this, we will be able to assess the extent, to which such system could operate on low-capacity devices such as wireless sensors and surveillance cameras.

Further, we plan to investigate the operation of the presented publish/subscribe paradigm for streams of time series data. This implies using different index techniques appropriate for solving the subsequence matching problem.

We plan to define a concept for Content-Based Routing of time series data based on the given matching semantics. This could be used in wireless sensor networks to minimize the propagation of sensed data and could lift the burden of signal processing and generating semantic events from sensed signals off these low-capacity devices.

## References

- [1] R. Agrawal, C. Faloutsos, and A. N. Swami. Efficient Similarity Search In Sequence Databases. In D. Lomet, editor, *FODO*, pages 69–84, Chicago, Illinois, 1993. Springer Verlag.
- [2] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R\*-tree: an efficient and robust access method for points and rectangles. In *1990 ACM SIGMOD*, pages 322–331. ACM Press, 1990.
- [3] A. Carzaniga, M. J. Rutherford, and A. L. Wolf. A routing scheme for content-based networking. In *Proceedings of IEEE INFOCOM 2004*, Hong Kong, China, Mar. 2004.
- [4] A. Carzaniga and A. L. Wolf. Forwarding in a content-based network. In *Proceedings of ACM SIGCOMM 2003*, pages 163–174, Karlsruhe, Germany, Aug. 2003.
- [5] C. Chatfield. *The Analysis of Time Series: An Introduction*. Chapman and Hall, London and New York, third edition, 1984.
- [6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, second edition, 2001.
- [7] R. Edwards and J. Magee. *Technical Analysis of Stock Trends*. American Management Association, 2001.
- [8] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.
- [9] D. Q. Goldin and P. C. Kanellakis. On similarity queries for time-series data: Constraint specification and implementation. In *Proceedings of the 1st International Conference on Principles and Practice of Constraint Programming (CP'95)*, Cassis, France, 1995. Springer Verlag.
- [10] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, Boston, Massachusetts, June 18-21 1984.
- [11] D. R. Jobman. *Handbook of Technical Analysis: A Comprehensive Guide to Analytical Methods, Trading Systems and Technical Indicators*. McGraw-Hill, 1994.
- [12] N. Katayama and S. Satoh. R\*-tree source code. Available at <http://research.nii.ac.jp/~katayama/homepage/research/srtree/>.
- [13] N. Katayama and S. Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. In J. Peckham, editor, *SIGMOD, May 13-15, 1997, Tucson, Arizona, USA*, pages 369–380. ACM Press, 1997.
- [14] R. Kurniawati, J. S. Jin, and J. Shepherd. SS+-tree: An improved index structure for similarity searches in a high-dimensional feature space. In *Storage and Retrieval for Image and Video Databases (SPIE)*, pages 110–120, 1997.
- [15] H. Liu and H. A. Jacobsen. Modelling uncertainties in publish/subscribe systems. In *ICDE 2004*, Boston, USA, March 2004.
- [16] S. K. Mitra. *Digital Signal Processing*. McGraw-Hill, 2001.
- [17] P. S. Modenov and A. S. Parkhomenko. *Geometric Transformations*. Academic Press, May 1966.
- [18] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. The grid file: an adaptable, symmetric multikey file structure. pages 582–598, 1988.
- [19] D. Raffei and A. O. Mendelzon. Efficient retrieval of similar time sequences using DFT. In *FODO*, pages 0–, 1998.
- [20] J. T. Robinson. The K-D-B-tree: a search structure for large multidimensional dynamic indexes. In *Proceedings of the 1981 ACM SIGMOD international conference on Management of data*, pages 10–18. ACM Press, 1981.
- [21] D. A. White and R. Jain. Similarity indexing: Algorithms and performance. In *Storage and Retrieval for Image and Video Databases (SPIE)*, pages 62–73, 1996.
- [22] D. A. White and R. Jain. Similarity indexing with the SS-tree. In *ICDE, February 26 - March 1, 1996, New Orleans, Louisiana*, pages 516–523. IEEE Computer Society, 1996.