NextPlace: A Spatio-temporal Prediction Framework for Pervasive Systems

Salvatore Scellato[1], Mirco Musolesi[2], Cecilia Mascolo[1], Vito Latora[3], and Andrew T. Campbell[4]

[1] Computer Laboratory, University of Cambridge, UK
[2] School of Computer Science, University of St. Andrews, UK
[3] Dipartimento di Fisica, University of Catania, Italy
[4] Department of Computer Science, Dartmouth College, USA

Abstract. Accurate and fine-grained prediction of future user location and geographical profile has interesting and promising applications including targeted content service, advertisement dissemination for mobile users, and recreational social networking tools for smart-phones. Existing techniques based on linear and probabilistic models are not able to provide accurate prediction of the location patterns from a spatio-temporal perspective, especially for long-term estimation. More specifically, they are able to only forecast the next location of a user, but not his/her *arrival time* and *residence time*, i.e., the interval of time spent in that location. Moreover, these techniques are often based on prediction models that are not able to extend predictions further in the future.

In this paper we present NextPlace, a novel approach to location prediction based on nonlinear time series analysis of the arrival and residence times of users in relevant places. NextPlace focuses on the predictability of single users when they visit their most important places, rather than on the transitions between different locations. We report about our evaluation using four different datasets and we compare our forecasting results to those obtained by means of the prediction techniques proposed in the literature. We show how we achieve higher performance compared to other predictors and also more stability over time, with an overall prediction precision of up to 90% and a performance increment of at least 50% with respect to the state of the art.

1 Introduction

The ability to predict future locations of people allows for a rich set of novel pervasive applications and systems: accurate content dissemination of location related information such as advertisement, leisure events reports and notifications [20, 1] could be implemented in a more effective way, avoiding the delivery of information to uninterested users, and, therefore providing, a better user experience. For example, by exploiting the availability of future location information, Web search engines such as Google, Bing or Yahoo! and location-based social network services such as Facebook Places and Foursquare may provide "location-aware" sponsored advertisements together with search results that are relevant to the predicted user movement patterns.

K. Lyons, J. Hightower, and E.M. Huang (Eds.): Pervasive 2011, LNCS 6696, pp. 152–169, 2011.

The increasing popularity of smart-phones equipped with GPS sensors makes location-aware computing a reality. Even in the case of devices where this information is not currently available, location can be roughly estimated by means of triangulation and cell estimation techniques or by profiling places through the analysis of the MAC addresses advertised by nearby devices and 802.11 access points [17]. In addition, these devices are increasingly always connected to the Internet, at least in areas where GPRS/EDGE or WiFi connectivity is present. Therefore, information about the current positions of users can be transmitted to a back-end server, where analysis of the data can be performed at run-time in order to predict future location patterns.

In this paper we propose NextPlace, a new prediction framework based on *nonlinear* time series analysis [12] for forecasting user behavior in different locations from a *spatio-temporal* point of view. NextPlace focuses on the temporal predictability of users presence when they visit their most important places. We do not focus on the transitions between different locations: instead, we focus on the estimation of the duration of a visit to a certain location and of the intervals between two subsequent visits. The existing techniques are able to forecast the next location of a user, but *not* his/her *arrival* and *residence time*, i.e., the interval of time spent in that location. Moreover, these techniques are often based on prediction models that are not able to extend predictions further in the future, since they mainly focus on the next movement of a user [19, 26, 23, 14, 2, 16].

We focus instead on patterns of residence in the set of locations that are more frequently visited by users. We show that, at least in the datasets under analysis, human presence in important places is characterized by a behavior that, even if at first glance seems apparently random, can be effectively captured by nonlinear models. Predictions are based on the collection of movement data that can be of different types: sets of GPS coordinates, registration patterns to access points or also information about presence in locations by means of passive and active transponders (such as badges). In addition, check-ins performed in location-based social networking services can be exploited to acquire movement data.

The proposed prediction technique consists of two steps. Firstly, we need to identify significant locations among which users move more frequently. Secondly, we apply a model able to predict user presence within these locations and relative residence time by means of techniques drawn from nonlinear time series analysis [12]. More specifically, the contribution of this paper can be summarized as follows:

- We describe NextPlace, a novel approach to user location prediction based on nonlinear time series analysis of visits that users pay to their most significant locations. NextPlace estimates the time of the future visits and expected residence time in those locations.
- We analyze four datasets of human movements: two GPS-based (representing respectively the positions of the users involved in the deployment of the CenceMe application at Dartmouth College [21] and the locations of cabs in San Francisco [24]) and two containing registration patterns of WiFi access points (at Dartmouth College [15] and within the Ile Sans Fils wireless network in Montreal, Canada [18]). We identify regularity and, more specifically, some previously uncaptured degree of determinism in patterns of user visits to their significant places by means of nonlinear analysis.

– We evaluate NextPlace comparing it with a probabilistic technique based on spatio-temporal Markov predictors [26] and with a linear model [6]. We report an overall prediction precision over the four datasets of up to 90%, with precision of up to 65% even after a number of hours, and a performance increment of at least 50% over Markov-based predictors. We show how the adoption of a nonlinear prediction framework can improve forecasting precision with respect to other techniques even for long-term predictions.

The rest of this paper is organized as follows: Section 2 describes NextPlace and its novel approach to prediction based on nonlinear time series analysis as well as illustrates the techniques we use for the extraction of significant places. Section 3 presents the implementation issues and the validation of our approach using real-world measurements, also reporting the results of the evaluation of our method against other predictors. Section 4 discusses related work and Section 5 concludes the paper illustrating potential future work.

2 Predicting Spatio-temporal Properties of Mobile Users

Any prediction of future user behavior is based on the assumption of determinism. From a practical point of view, determinism simply means that future events are determined by past events, so that every time a particular configuration or situation is observed, the same (or a similar) outcome will follow. Since in human societies daily and weekly routines are well-established, human activities are characterized by a certain degree of regularity and predictability [8].

The intuition behind NextPlace is that the sequence of important locations that an individual visits every day is more or less fixed, with only minor variations that are also usually deterministically defined. As an example, if a woman periodically goes to the gym on Mondays and Thursdays, she may change her routine for those days, but the changed routine will be more or less the same over different weeks. Therefore, the sequence of events may still be predictable.

From a formal point of view, let us consider a certain number of mobile users, where user i freely moves among different locations. For the moment, we do not explicitly focus on how these locations can be identified, and only assume that the start time and the duration of each visit of a user to a given location can be determined. A visit of a user is simply defined by the tuple (u, l, t, d), where t and d are respectively the time of arrival and the residence time of user u in location l. It is worth noting that this approach does not model movements but, rather, residence time in some locations, hence, it can also be adopted in systems without any spatial or geographical information about locations, i.e., access points in 802.11 WLANs.

We now introduce the two steps of NextPlace and the basic theory behind them. We first describe how we isolate the user's significant places, exploiting the technique proposed by Kim et al. in [14]. Then, we describe our novel method for the estimation of future times of arrival and residence times in the different significant places and how we exploit this prediction to compute accurate estimation of where the user will be after a given time interval. Finally, we describe the mathematical details of the prediction techniques behind our approach.

(a) (b)

Fig. 1. Example of frequency map using GPS traces. Higher peaks in (a) reveal places where user spent most of their time and which represent its significant locations: in (b) we show some visits to these significant places reported on a geographical map.

extracting places and timestamps from MOVES

2.1 Significant Places Extraction

In this section we present two methods we use to extract significant locations from both GPS information and WiFi association logs, the two most commonly available sources of data about user movements.

Extracting Places from GPS Data. Many solutions for the extraction of significant places from GPS measurements have been presented in the literature [11, 28, 2]. We choose one that is based on the residence time of a user to quantify the importance of a place for him/her: the intuition being that permanence at a place is directly proportional to the importance that is attributed to it by the user.

As proposed in [14], we apply a 2-D Gaussian distribution weighted by the residence time at each GPS point. This means that at each point the Gaussian distribution uniformly contributes also to nearby points, smoothing out values that are close together. The value of the variance for the Gaussian distributions that we choose is $\sigma = 10$ meters, which is related to the average GPS accuracy[1]. The resulting *frequency map* contains peaks which give information about the position of popular locations: we consider regions that are above a certain threshold T as significant places. The threshold T can be chosen as a fraction of the maximum value of the frequency map. We will show the application of this technique and how the value of the threshold T can be selected using two GPS-based datasets in Section 3.

In Figure 1(a) a close-up of a frequency map is shown: when a threshold is applied to the map, only higher peaks are selected and each peak generates an area defined

[1] `http://www.gps.gov/`

by a continuous boundary. All GPS points within that area result in visits to the same significant place. As an example, if we choose a threshold equal to 15% of the highest peak of the map, we obtain the visits to significant places shown over the area map in Figure 1(b).

Extracting Places from WiFi Logs. Alternatively, we can derive significant places from user registrations to 802.11 access points. Since these access points are fixed and easily identifiable from their globally unique MAC address, this information can be exploited to extract visit patterns to a set of locations in a straightforward manner. From this point of view, the most frequently seen access points are natural candidates to represent significant places. Hence, we can define as popular places for a user the access points he/she connects to more often, providing that a sufficient number of visits has been recorded to a given access point. More specifically, we define an access point as a significant place for a certain user if this user has a sequence of at least n visits to the access point, in order to filter out all the access points that are seldom visited and to have a sufficient number of observations from a statistical point of view. For the analysis presented in this paper, we select n equal to 20.

2.2 Predicting User Behavior

We now describe NextPlace's location prediction algorithm: in order to obtain an estimation of the future behavior, the history of visits of a user to each of its significant locations is considered. Then, for each location we try to predict when the next visits will take place and for how long they will last. After this estimation, the predictions obtained for different locations is analyzed, in order to produce a unique prediction of where the user will be at a given future instant of time. A theoretical foundation of this technique is described in Section 2.3.

For each user we keep track of all previous visits to a set of locations, that is, for each visit we consider the instant when it started and how long it lasted. The algorithm predicts the next visits to a given location by means of the previous history of visits $((t_1, d_1), (t_2, d_2), \ldots, (t_n, d_n))$:

1. two time series are created from the sequence of previous visits: the time series of the visit daily start times C and the time series of the visit durations D defined as follows:
$$C = (c_1, c_2, \ldots, c_n)$$
$$D = (d_1, d_2, \ldots, d_n)$$
where c_i is the time of the day in seconds corresponding to the time instant t_i (i.e., c_i is in the range $[0, 86400]$);
2. we search in the time series C sequences of m consecutive values $(c_{i-m+1}, \ldots, c_i)$ that are closely similar to the last m values $(c_{n-m+1}, \ldots, c_n)$[2];
3. the next value of time series C is estimated by averaging all the values c_{i+1} that follow each found sequence;

[2] We will discuss the choice of parameter m in the next section.

consider also a weekly approach

4. at the same time, in time series D the corresponding sequences $(d_{i-m+1}, \ldots, d_i)$ are selected; the sequences have to be located exactly at the same indexes as those in C;
5. the next value of time series D is then estimated by averaging all the values d_{i+1} that follow these sequences.

As an example, if the last three visits of a certain user to a location are Monday at 6:30pm, Monday at 10:00pm and Tuesday at 8:15am, we analyze the history of visits in order to find sequences that are numerically close to (6:30pm, 10:00pm, 8:15am), i.e. (6:10pm, 9:50pm, 8:35am) and (6:35pm, 10:10pm, 8:00am): then, assuming that the next visits that follow these subsequences start at 1:10pm and 12:40pm and last for 40 and 30 minutes respectively, we estimate the next visit at 12:55pm for 35 minutes, averaging both arrival times and duration times.

The main idea behind this algorithm is the assumption that human behavior is strongly determined by daily patterns: the sequence of visit start times is therefore mapped to a 24-hour time interval, focusing only on the start time of each visit. The choice of the value m has an impact on the accuracy of the prediction: in fact, this can be improved by taking into account more visits in order to identify particular patterns that may be present only in certain intervals of time such as specific days.

We can generalize this algorithm to predict not only the next visit to a location, but also successive visits in the future: in fact, we can choose to average together not only the next values of each subsequences but also values that are 2 or more steps ahead. However, the prediction of time series can become inaccurate when adopted to calculate further values in the future [12].

parameter m

Since we can predict when the future visits to all significant locations will start and for how long they will last, we can design a simple method to predict the location where the user will be at a given time in the future. Let us suppose that at time T we want to predict in which significant location user i will be after ΔT seconds. Then, the following steps are performed:

1. for each location the sequence of the next k visits (starting with $k = 1$) are predicted and a global sequence of all predicted visits $(loc_1, t_1, d_1), \ldots, (loc_n, t_n, d_n)$ is created, with $t_1 \leq \cdots \leq t_n$;
2. if there is a prediction (loc_i, t_i, d_i) which satisfies $t_i \leq T + \Delta T \leq t_i + d_i$, then loc_i is returned as predicted location (in case several predictions exist which satisfy the predicate, we choose at random between them);
3. if no prediction satisfies the condition stated above, there are two cases: if the minimum start time t_1 of the current predicted visits is smaller than $T + \Delta T$, then prediction needs to be extended further in the future in order to find a suitable visit, thus the parameter k is doubled and the algorithm is repeated considering new predicted visits. Otherwise, extending the prediction provides visits which start after $T + \Delta T$ and which cannot be exploited for prediction: thus, the algorithm terminates returning that the user will not be in any significant location.

Note that it is realistic for a user to be predicted as being outside the set of significant places (e.g., maybe transitioning from one to another) and that our technique is also able to predict this state.

2.3 Nonlinear Prediction Framework: Key Concepts and Practical Implementation Issues

In this section we provide a brief overview of the key concepts at the basis of the forecasting framework and we discuss the practical issues in implementing it.

In this work we adopt a prediction technique inspired by *nonlinear time series analysis* [12]. A time series can be seen as a collection of scalar observations of a given system made sequentially in time and spaced at uniform time intervals, albeit this last assumption can be relaxed to allow any kind of temporal measurement pattern [6].

While the scalar sequence of values contained in a time series may appear completely unrelated to the underlying system, it is possible to uncover the characteristics of its dynamic evolution by analyzing sub-sequences of the time series itself. In order to investigate the structure of the original system, the time series values must be transformed in a sequence of vectors with a technique called *delay embedding*.

More formally, a time series $(s_0, s_1, \ldots, s_N)$ can be embedded in a m-dimensional space by defining an appropriate delay ν and then creating a *delay vector reconstruction* for the time series value s_n as follows:

$$\beta_n = [s_{n-(m-1)\nu}, s_{n-(m-2)\nu}, \ldots, s_{n-\nu}, s_n]$$

where all vectors β_n have m components and are defined in a so called *embedding space*. Note that m is the parameter used in the algorithm described in Section 2.2.

The values of the parameters m and ν greatly affect the accuracy of the representation. Nonetheless, a fundamental mathematical result (the so-called *embedding theorem* [12]) ensures that a suitable value for m does exist and is related to the complexity of the underlying system. At the same time, ν might be chosen to represent a suitable time scale of the phenomenon, since consecutive values in the time series should not be too strongly correlated to each other.

An effective predictive model can be generated directly from time series data through the delay embedding. Let us suppose that a prediction for the value $s_{N+\Delta n}$, a time Δn ahead of N, must be made for the time series $(s_0, s_1, \ldots, s_N)$. The steps of the prediction process are as follows:

1. The time series is embedded in a m-dimensional space by defining an appropriate time delay ν and then creating the related embedding space;
2. The embedding space is searched for all the vectors that are close, with respect to some given metric distance, to vector β_N: more formally, a neighborhood $U_\epsilon(\beta_N)$ of radius ϵ around the vector β_N is created;
3. Since determinism involves that future events are set causally by past events, and since all vectors $\beta_n \in U_\epsilon(\beta_N)$ describe past events similar to the past events of β_N, the prediction $p_{N+\Delta n}$ is taken as the average of all the values $s_{n+\Delta n}$

$$p_{N+\Delta n} = \frac{1}{|U_\epsilon(\beta_N)|} \sum_{\beta_n \in U_\epsilon(\beta_N)} s_{n+\Delta n}$$

where $|U_\epsilon(\beta_N)|$ denotes the number of elements of the neighborhood $U_\epsilon(\beta_N)$. The value of ϵ should be chosen in order to obtain a sufficient number of vectors for the prediction.

Intuitively, this algorithm searches the past history to find sequences of values that are very similar to the recent history: assuming that the evolution is ruled by deterministic patterns, a given state will always be followed by the same outcome.

In our implementation we have chosen $\nu = 1$, since we do not have to deal with particular time scales which require to skip some values of our time series. As suggested in [12], the radius ϵ of the vector neighborhood is chosen in order to be 10% of the standard deviation of each time series: this value allows us to obtain enough vectors to perform prediction and, at the same time, filters out vectors that are not close to β_N.

We note that for each prediction all vectors in the embedding space have to be considered and searched. For this reason, it is wise to use an efficient method to find nearest neighbors in the embedding space: the main computational burden is the calculation of the neighborhood $U_\epsilon(\beta_N)$ and the asymptotic complexity $O(N^2)$ can be reduced to $O(N \log N)$ with binary trees or even to $O(N)$ with a box-assisted search algorithm [25], which is the method we implement.

3 Validation of the Prediction Framework Using Real-World Measurements

In this section we introduce the datasets used in our analysis and we describe how we process them in order to extract significant places. Then, we investigate the predictability of the time series extracted from sequences of visits of each user to his/her significant locations, using standard metrics adopted in time series analysis. Finally, we compare NextPlace prediction performance against other prediction methods.

3.1 Datasets

For the evaluation of our approach we choose four different datasets of human movements:

1. **Cabspotting.** This dataset is composed of movement traces of taxi cabs in San Francisco, USA, with GPS coordinates of approximately 500 taxis collected over 30 days in the San Francisco Bay Area. Each vehicle is equipped with a GPS tracking device that is used by dispatchers to efficiently reach customers [24]. The average time interval between two consecutive GPS measurements is less than 60 seconds.
2. **CenceMe GPS.** This dataset was collected during the deployment of CenceMe [21], a system for recreational personal sensing, at Dartmouth College. The GPS data was collected by means of 20 Nokia N95 phones carried by postgraduate students and staff members from the Department of Computer Science and the Department of Biology.
3. **Dartmouth WiFi.** This dataset was extracted from the SNMP logs of the WiFi LAN of the Dartmouth College campus. The compact nature of the campus means that the signal range of interior APs extends to most of the campus outdoor areas. Between 2001 and 2004 data about traffic in the access points was collected through three techniques: syslog events, SNMP polls, and network sniffers [9, 15].

Table 1. Properties of the different datasets: total number of users N, total number of visits V, total number of significant places P, average number of significant places per user p, average number of visits per user v, average residence time in a place D (seconds), total trace length and average proportion of time spent by each user in significant places

Dataset	N	V	P	p	v	D [s]	Trace length	Significant time
Cabspotting	252	150612	6122	24.29	597	231	23 days	7.27%
CenceMe GPS	19	3832	225	11.84	201	696	12 days	14.74%
Dartmouth WiFi	2043	772217	539	17.87	377	2094	60 days	11.24%
Ile Sans Fils	804	142407	173	3.61	177	5296	370 days	0.18%

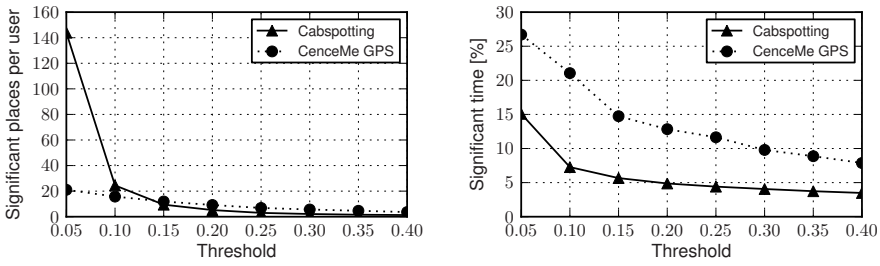
4. **Ile Sans Fils.** Ile Sans Fils [18] is a non-profit organization which operates a network of free WiFi hotspots in Montreal, Canada. It now counts over 45,000 users with 140 hotspots located in publicly accessible spaces. These hotspots are deployed mostly in cafes, restaurants and bars, libraries, but also outdoor to cover parks and sections of popular commercial streets.

We choose a subset of regularly active users for each original dataset, filtering out all the users that appear only a few times and for which prediction may be worthless. In Table 1 we report some important characteristics and metrics of the resulting datasets.

3.2 Practical Issues

In order to extract significant places for each user in the Cabspotting and CenceMe GPS datasets, which are composed of GPS measurements, we need to choose a suitable threshold T for the frequency map. Thus, we investigate how the average number of significant places per user changes as a function of the threshold itself. As reported in Figure 2(a), the average number of places decreases as the threshold increases: for the Cabspotting dataset a suitable choice is $T = 0.10$, where the curve changes its slope, which denotes the transition from a situation with many unimportant significant areas to a situation with less but probably more important places. However, in the case of the CenceMe GPS dataset such transition does not occur: hence, we investigate how the percentage of time spent in significant locations changes with T, as reported in Figure 2(b): this percentage quickly decreases with T but the steepness of the curve changes at $T = 0.15$. Hence, we choose the value of $T = 0.15$ for this dataset. These values of T result in an average number of about 24 and 12 places per user for the Cabspotting and CenceMe GPS datasets, respectively.

When dealing with GPS measurements, the duration of a visit can be computed as the difference between two consecutive GPS samples. However, the GPS measurement process usually involves a periodic sampling of the location. When the user is located for a long time interval inside the same region, this results in several successive short visits being recorded, whose length depends on the adopted sampling interval. The same problem may occur with WiFi association logs: since WiFi connectivity may be intermittently available and handoff mechanisms are in place in this type of network infrastructure, a long residence time may be split in several shorter sessions.



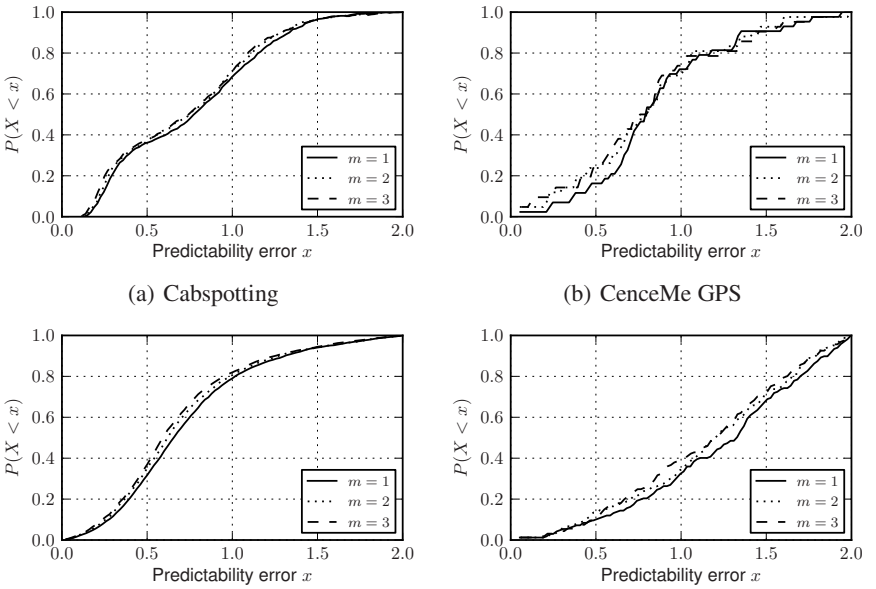
(a) (b)

Fig. 2. Average number of significant places per user (a) and percentage of time spent in significant locations (b) as a function of the threshold T of the frequency map for the Cabspotting dataset and the CenceMe GPS dataset

In order to infer a more accurate residence time of the user in a certain region, we apply a merging procedure to the dataset of the sequence of visits. Given a sequence of visits to the same location $(t_1, d_1), (t_2, d_2), \ldots (t_n, d_n)$, if the end time of a visit is close to the start time of the next one, that is if $t_{i+1} - (t_i + d_i) \leq \delta$, we merge them in a new visit starting at t_i and ending at $t_{i+1} + d_{i+1}$. In this way the visits obtained are more likely to mimic the real patterns of presence of users, thus improving prediction. We adopted the value of $\delta = 60$ seconds for the Cabspotting dataset and $\delta = 180$ seconds for the CenceMe GPS dataset, since these are the values of the scanning period for the GPS data acquisition. On the other hand, we apply the same merging procedure to WiFi association logs in the Dartmouth and Ile Sans Fils datasets with a value of $\delta = 300$ seconds, in order to filter out casual disconnections from the access point which may last for few minutes.

From a statistical point of view, these datasets show different characteristics, as reported in Table 1: while Cabspotting, Dartmouth WiFi and Ile Sans Fils contain measurements for hundreds or thousands of users, CenceMe GPS consists of data related to a smaller group of moving users. On average about 12 significant locations have been recorded for each user in the CenceMe GPS dataset. In the Dartmouth WiFi and Cabspotting datasets the number of significant places is 18 and 24, respectively. On the other hand, in the Ile Sans Fils dataset we have less than 4 significant locations per user. This is due to the fact that the Ile Sans Fils dataset contains association logs with access points located in public spaces, thus, a large portion of individuals are seen just in few locations. In fact, public access point are not likely to capture some important places for a given user, such as his/her home and working place. There are also differences in the residence time of users in their significant locations: while for Ile Sans Fils and Dartmouth WiFi the average residence time is about 90 and 30 minutes, in the Cabspotting and CenceMe GPS datasets it is about 5 and 10 minutes.

Finally, the amount of time spent in significant locations is crucial to the investigation of the performance of the location prediction technique. While in the CenceMe GPS and in the Dartmouth WiFi datasets each user spends on average 14.74% and 11.24% of their time in a significant location, this value drops to 7.27% in the Cabspotting dataset



(a) Cabspotting (b) CenceMe GPS

(c) Dartmouth WiFi (d) Ile Sans Fils

Fig. 3. Cumulative Distribution Function of the predictability error of the time series of the start instants extracted from the various datasets. We report the results for different values of the embedding dimension m adopted in the prediction method.

and to 0.18% in the Ile Sans Fils dataset, since it covers a longer period of time (more than one year) and many of its users are present less regularly than in the other datasets.

3.3 Time Series Predictability Test

In order to exploit time series techniques to predict user behavior, we first need to investigate if determinism is present in the extracted time series. In other words, we want to evaluate the *predictability* of these time series.

Let us consider a time series $(s_0, s_1, \ldots, s_N)$. If a real measurement for s_{N+1} is given, the prediction error is the difference between s_{N+1} and the predicted value p_{N+1}. Given a prediction technique, it is possible to obtain predicted values $(p_0, p_1, \ldots, p_N)$ for the whole time series. Then, the *mean quadratic prediction error* can be evaluated as $\varepsilon = \frac{1}{N} \sum_{n=1}^{N} (s_n - p_n)^2$. Large values of ε indicate that the prediction is not accurate and the time series is not predictable.

The evaluation of ε is based on the comparison to the variance σ^2 of the time series: thus, a convenient way of deciding whether ε is small or large is to take the ratio $\frac{\varepsilon}{\sigma^2}$, which is the *predictability error*: if this ratio is close to 1, then, the mean quadratic prediction error is large, while if it is close to 0, the mean quadratic prediction error is small. We refer to this ratio as the *predictability error* of a prediction algorithm. The absolute error value ε may be meaningless if not compared to the average amount of

fluctuations a time series exhibits: by dividing by the variance of the series we can normalize the error and compare the prediction accuracy for different time series.

We exploit this metric to evaluate whether the time series extracted from user visits in the different datasets are predictable. We divide each dataset in two halves: we use the first half to build the model and we compute predicted values of the second half and vice versa. A value equal to 1 means that no determinism is present in the time series, since in this case the predictor has the same accuracy of the simple average value, whereas a value closer to 0 indicates a high degree of determinism.

In Figure 3 we show the Cumulative Distribution Function of the predictability error for the time series of the visit start times for different values of the embedding dimension m. We have also investigated the predictability error for the time series of visit end times, obtaining similar results, which we do not show due to space limitations. On average, a large proportion of users exhibit predictability: in the Dartmouth WiFi dataset 80% of the time series show predictability error smaller than 1, whereas in the CenceMe GPS and Cabspotting datasets the same figure is 70% and it drops to 40% in the Ile Sans Fils dataset, which show less predictability than the others. This is due to the fact that visits may not occur every day with the same pattern for access points in public places, since different individuals are likely to show less regularity in public space than in more personal locations as living or working places, which are not present in this dataset. Moreover, in all datasets the predictability error is lower for higher values of the embedding dimension m: this confirms that nonlinear methods improve prediction quality, since they are able to capture and recognise specific patterns of visits and to estimate when the next visit will be. However, we have noticed that values of $m \geq 4$ show worse performance because we do not have sufficient statistics in order to make a correct prediction.

Interestingly, we expected to observe a lower degree of regularity in the Cabspotting traces, since the movements of a taxi are related to the destinations of the different customers and these destinations can be hardly predictable. Nonetheless, we were able to identify a set of places among which taxis move with more regular patterns. These places correspond to areas where taxi drivers periodically go and wait for new customers, such as touristic locations, shopping malls, cinemas, and they tend to exhibit regular and predictable patterns.

3.4 Evaluating Prediction Accuracy

We compare the performance of NextPlace with those of other two methods: a state-of-the-art Markov-based spatio-temporal predictor and a modified version of NextPlace, where time series of visits are predicted with linear methods rather than with nonlinear algorithms.

Methodology. Firstly, we compare NextPlace with a more sophisticated *spatio-temporal Markov predictor* derived by extending the techniques presented in [26]. To the best of our knowledge, this is the most accurate algorithm that has been presented in the literature for this class of prediction problems, because it combines spatial and temporal dimensions to estimate both next location and handover time for users in a cellular network.

Consider a user visit history among several locations $H = (t_1, d_1, l_1), \ldots, (t_n, d_n, l_n)$, where t_i is the time when the user arrived at location l_i and d_i is the residence time in that location. Then, from H we extract the location history $L = l_1, \ldots, l_n$ and the order-k location context $L_k = L(n-k+1, n) = l_{n-k+1}, \ldots, l_{n-1}, l_n$. The history L is searched for instances of the context L_k and, for each destination that follows an instance, we examine the duration of the previous residence time. More formally, we extract the following set of inter-arrival times A_x and set of durations D_x for each possible destination x:

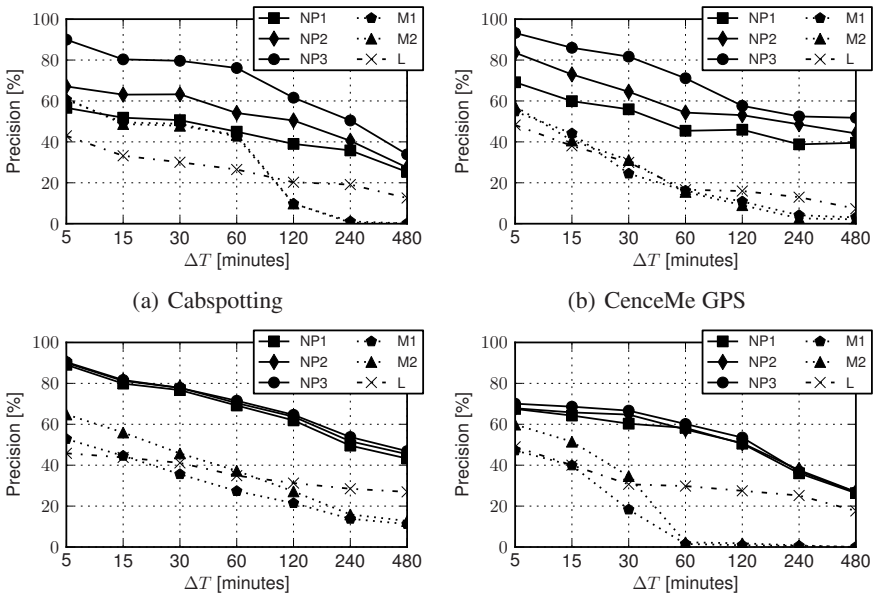
$$A_x = \{t_{i+1} - t_i \quad \text{if} \quad L(i-k+1, i+1) = (L_k, x)\}$$
$$D_x = \{d_{i+1} \quad \text{if} \quad L(i-k+1, i+1) = (L_k, x)\}$$

Then, we compute the estimated time when the user will move to location x and the estimated residence time in x by using a CDF predictor with probability $p = 0.8$ [26]. Moreover, a Markov predictor of order k is used to assign the probability of transition between the current location and the possible destinations. Finally, spatial and temporal information are combined to obtain the predicted location. In order to predict not only the next location but also the subsequent ones, we extend this approach taking the predicted location as the current one and computing again the next movement. We refer the interested reader to the original paper for further details [26].

To understand how largely NextPlace relies on the performance of the nonlinear time series predictor, we can design a linear version of our prediction technique. We use an *order-k running average predictor* instead of a nonlinear method to estimate the future values of a time series: given the sequence of previous visits of a user to a location, the last k visit duration times and k intervals between visits are averaged to obtain a prediction of future visits. Then, the future location is chosen among several predicted locations according to the same algorithm at the basis of the nonlinear predictor (presented in Section 2.2). However, this simplistic time series predictor ignores how user behavior changes over time, since high heterogeneity can be observed in visits occurring during different times of the day. Focusing only on recent data and not investigating these temporal aspects may not be sufficient to obtain accurate estimates.

Results. We now evaluate the performance of NextPlace with the nonlinear predictor presented in Section 2.2 compared to the other predictors previously described.

We use the following definition of correctness: if we predict, at time T, that the user i will be at location l at time $T_P = T + \Delta T$, the prediction is considered correct only if the user is at l at any time during the interval $[T_P - \theta, T_P + \theta]$, where θ is the error margin. It is important to note that each prediction algorithm can also estimate if the user will not be in any of her significant places: thus, a prediction may be correct whether the user is predicted to be in a particular location l and then he/she is in l or if the user is predicted not to be in any significant location and then, in fact, she is not. However, as reported in Table 1, the fraction of time that on average users spend in their significant locations ranges between 14.74% in the CenceMe GPS dataset and only 0.18% in the Ile Sans Fils dataset. Hence, it is not easy to understand if predictions are accurate because a method is performing well or because, on average, it is just easier to predict the user outside of all her significant locations.



(a) Cabspotting (b) CenceMe GPS

(c) Dartmouth WiFi (d) Ile Sans Fils

Fig. 4. Prediction precision as a function of time interval ΔT for the different datasets and for different predictors: NextPlace with nonlinear predictor for different values of embedding dimension $m = 1, 2, 3$ (NL1-NL2-NL3), first-order and second-order Markov-based (M1-M2) and NextPlace with linear predictor (L). Error margin is $\theta = 900$ seconds.

Therefore, we introduce an accuracy metric that takes into account this issue. We define the *prediction precision* as the ratio between the number of correct predictions and the number of all attempted predictions which forecast the user to be in a significant location. We do not consider for the evaluation any predictions which forecast the user outside her significant locations.

We report the performance of different predictors: we test NextPlace with different values of the embedding parameter $m = 1, 2, 3$, two order-1 and order-2 Markov-based predictors and the linear version of NextPlace with a running average predictor considering the last $m = 4$ values. For each dataset, we use the first half to build a prediction model and then we compute predictions during the second half and, for each user, we make 1000 predictions at uniformly distributed random instants. Finally, prediction precision is computed and we investigate how it changes with ΔT, using an error margin $\theta = 900$ seconds. All results are averaged over 20 runs with different random seeds.

We see in Figure 4 that for all datasets, NextPlace with its nonlinear predictor is always outperforming the other methods. We also note that using a higher value of m improves prediction quality, as it can be appreciated especially in the GPS-based Cabspotting and CenceMe GPS datasets. Similarly, Markov models are able to provide correct predictions when ΔT is smaller than 1 hour: however, except for the Ile Sans Fils dataset, the performance of the nonlinear NextPlace are at least about 50% better

of the Markov-based predictors, since they reach a maximum precision of 60% while NextPlace achieves a precision higher than 90%. Moreover, when ΔT increases, the precision of Markov predictors decreases rapidly and the performance gap with the nonlinear approach widens. This can be explained by the fact that Markov predictors are generally employed to predict the next movement and, thus, when predictions are extended in the future, movement after movement, a large error is accumulated.

If we substitute the nonlinear predictor in NextPlace with a linear one, we observe a similar trend but precision is considerably lower, since errors on time series prediction are larger and, hence, affect the location prediction. However, NextPlace with both nonlinear and linear predictors is less dependent on ΔT than Markov models, which show a lower precision when predictions are extended in the future. Again, this demonstrates how NextPlace, which focuses only on temporal information of visits in significant places, is more robust for long-term predictions.

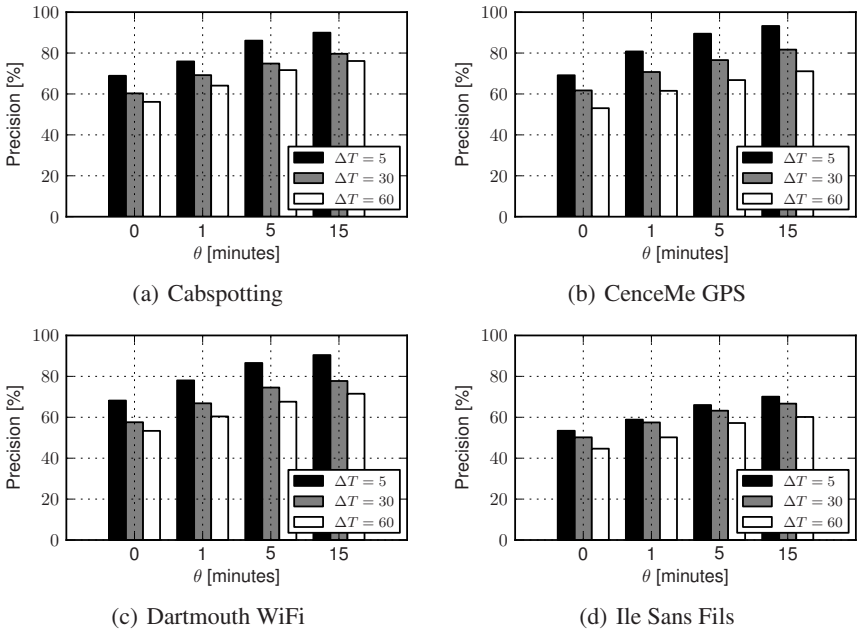
As discussed in Section 3, the Ile Sans Fils dataset exhibits less predictability. This is confirmed by the analysis of prediction precision, which shows the lowest figures among all the datasets. The other datasets score a precision equal to about 90% for $\Delta T = 5$ minutes and around 70% for $\Delta T = 60$ minutes. We also investigate the impact of the error margin θ on prediction results: prediction precision is lower for smaller error margins, but it shows the same trends for all predictors and for all the datasets. In Figure 5 we report how prediction precision of our nonlinear approach with $m = 3$ is affected by different error margins for some values of ΔT. Even with $\theta = 0$, which represents the worst case scenario, prediction precision is between 50% and 60% after $\Delta T = 60$ minutes for all datasets except Ile Sans Fils, where it is below 50%.

From a general point of view, our evaluation shows how NextPlace achieves high prediction accuracy, even for long-term predictions made some hours in advance. Furthermore, these results also show how focusing on spatial movements, as Markov models do, may be useful only for short-term predictions. Instead, focusing just on temporal information about recurrent patterns in significant places proves to be more robust both for short-term and long-term predictions, since NextPlace outperforms Markov models even for small values of ΔT.

4 Related Work

Pioneering work [3,4] has focused on the analysis of mobility traces in order to gain insight about human mobility patterns. Key papers in this area include studies on mobility and connectivity patterns, such as [5, 13]. The main findings are that contact duration and inter-contacts time between individuals can be represented by means of power-law distributions and that these patterns may be used to develop more efficient opportunistic protocols [10]. In addition, temporal rhythms of human behavior have been studied and modeled to discover daily activity patterns, to infer relationships and to determine significant locations [7]. This related body of work concentrates on the *statistical characterization* of temporal behavioral patterns of groups of users, whereas we concentrate on prediction of single users.

The evaluation of prediction techniques applied to the problem of forecasting the next location (but not the arrival time to that location and the corresponding residence



(a) Cabspotting (b) CenceMe GPS

(c) Dartmouth WiFi (d) Ile Sans Fils

Fig. 5. Prediction precision of NextPlace with nonlinear predictor with $m = 3$ as a function of error margin θ and for different values of ΔT

time) are presented in [27]. A prediction framework based on spatio-temporal patterns in collective mobility trajectories has been presented in [22]: this method attempts to predict the next location of a moving object by matching a new trajectory to a corpus of global frequent ones. While this prediction technique is more general, as it captures dependencies between visits at different places, our method includes time-of-day information and does not rely on global patterns, allowing prediction to be made also for users who deviate from collective behavior. In [2] the authors present a model of user location prediction from GPS data. A simple first-order Markov model to predict the transitions between significant places is used, albeit in this work temporal aspects are not taken into consideration. In [19] the significant places are extracted by means of a discriminative relational Markov network; then, a generative dynamic Bayesian network is used to learn transportation routines. Another system for the prediction of future network connectivity based on a second-order Markov model is BreadCrumbs [23]. Again, this system is able to predict only the next location of the user and not the time of the transitions and the interval of time during which users reside in that specific location. Similarly, Markov based techniques have also been applied to the prediction of the destinations (geographical locations) of vehicles using for example partial trajectories [16]. As we have shown in the evaluation section, this class of models is able to provide precise predictions only for instants of time close in the future, given the inherent memorylessness of Markov predictors.

5 Conclusions

In this paper we have presented NextPlace, a new approach to spatio-temporal user location prediction based on nonlinear analysis of the time series of start times and duration times of visits to significant locations. To the best to our knowledge, this is the first approach that not only allows to forecast the next location of a user, but also his/her *arrival* and *residence time*, i.e., the interval of time spent in that location. Moreover, existing models are not able to extend predictions further in the future, since they mainly focus on the next movement of a user.

We have evaluated NextPlace comparing it with a version based on a linear predictor and a probabilistic technique based on spatio-temporal Markov predictors over four different datasets. We have reported an overall prediction precision up to 90% and a performance increment of at least 50% over the state of the art. We have showed how the adoption of a nonlinear prediction framework can improve prediction precision with respect to other techniques even for long-term predictions.

As future work, there is a number of potential improvements that can be pursued. Regular collective human rhythms can be exploited to refine the prediction and a probabilistic framework can be used to choose between equally promising next locations, giving more flexibility to applications. Finally, we are interested in the investigation of prediction models which take into account human rhythms on a weekly basis, in order to better capture regular human behavior on a longer time scale.

References

1. Aalto, L., Göthlin, N., Korhonen, J., Ojala, T.: Bluetooth and WAP Push Based Location-aware Mobile Advertising System. In: Proceedings of MobiSys 2004, pp. 49–58 (2004)
2. Ashbrook, D., Starner, T.: Using GPS to Learn Significant Locations and Predict Movement Across Multiple Users. Journal of Personal and Ubiquitous Computing 7(5), 275–286 (2003)
3. Balachandran, A., Voelker, G.M., Bahl, P., Rangan, P.V.: Characterizing User Behavior and Network Performance in a Public Wireless LAN. In: Proceedings of SIGMETRICS 2002 (2002)
4. Balazinska, M., Castro, P.: Characterizing Mobility and Network Usage in a Corporate Wireless Local-Area Network. In: Proceedings of MobiSys 2003, San Francisco, CA (May 2003)
5. Chaintreau, A., Hui, P., Crowcroft, J., Diot, C., Gass, R., Scott, J.: Impact of Human Mobility on Opportunistic Forwarding Algorithms. IEEE Transactions on Mobile Computing 6(6), 606–620 (2007)
6. Chatfield, C.: The Analysis of Time Series: An Introduction, 5th edn. Chapman & Hall/CRC, London (July 1995)
7. Eagle, N., Pentland, A.S.: Reality Mining: Sensing Complex Social Systems. Personal Ubiquitous Comput. 10(4), 255–268 (2006)
8. Gonzalez, M.C., Hidalgo, C.A., Barabasi, A.-L.: Understanding Individual Human Mobility Patterns. Nature 453(7196), 779–782 (2008)
9. Henderson, T., Kotz, D., Abyzov, I.: The Changing Usage of a Mature Campus-wide Wireless Network. In: Proceedings of MobiCom 2004, New York, NY, USA, pp. 187–201 (2004)
10. Jain, S., Fall, K., Patra, R.: Routing in a Delay Tolerant Network. In: Proceedings of SIGCOMM 2004 (2004)
11. Kang, J.H., Welbourne, W., Stewart, B., Borriello, G.: Extracting Places from Traces of Locations. SIGMOBILE Mobile Computing Communication Review 9(3), 58–68 (2005)

12. Kantz, H., Schreiber, T.: Nonlinear Time Series Analysis. Cambridge University Press, Cambridge (2004)
13. Karagiannis, T., Le Boudec, J.-Y., Vojnovic, M.: Power Law and Exponential Decay of Intercontact Times Between Mobile Devices. In: Proceedings of MobiCom 2007, pp. 183–194 (2007)
14. Kim, M., Kotz, D., Kim, S.: Extracting a Mobility Model from Real User Traces. In: Proceedings of INFOCOM 2006 (April 2006)
15. Kotz, D., Henderson, T., Abyzov, I.: CRAWDAD trace dartmouth/campus/movement/01_04 (v. 2005-03-08) (March 2005), http://crawdad.cs.dartmouth.edu/
16. Krumm, J., Horvitz, E.: Predestination: Inferring Destinations from Partial Trajectories. In: Dourish, P., Friday, A. (eds.) UbiComp 2006. LNCS, vol. 4206, pp. 243–260. Springer, Heidelberg (2006)
17. LaMarca, A., Chawathe, Y., Consolvo, S., Hightower, J., Smith, I., Scott, J., Sohn, T., Howard, J., Hughes, J., Potter, F., Tabert, J., Powledge, P., Borriello, G., Schilit, B.: Place Lab: Device Positioning Using Radio Beacons in the Wild. In: Gellersen, H.-W., Want, R., Schmidt, A. (eds.) PERVASIVE 2005. LNCS, vol. 3468, pp. 116–133. Springer, Heidelberg (2005)
18. Lenczner, M., Gregoire, B., Roulx, F.: CRAWDAD data set ilesansfil/wifidog (v. 2007-08-27) (August 2007), http://www.crawdad.cs.dartmouth.edu/ilesansfil/wifidog
19. Liao, L., Patterson, D.J., Fox, D., Kautz, H.: Building Personal Maps from GPS Data. In: Proceedings of IJCAI Workshop on Modeling Others from Observation (2005)
20. Marmasse, N., Schmandt, C.: Location-Aware Information Delivery with ComMotion. In: Thomas, P., Gellersen, H.-W. (eds.) HUC 2000. LNCS, vol. 1927, pp. 157–171. Springer, Heidelberg (2000)
21. Miluzzo, E., Lane, N.D., Fodor, K., Peterson, R., Lu, H., Musolesi, M., Eisenman, S.B., Zheng, X., Campbell, A.T.: Sensing Meets Mobile Social Networks: the Design, Implementation and Evaluation of the CenceMe Application. In: Proceedings of SenSys 2008, pp. 337–350. ACM, New York (2008)
22. Monreale, A., Pinelli, F., Trasarti, R., Giannotti, F.: WhereNext: a location predictor on trajectory pattern mining. In: Proceedings of SIGKDD 2009, pp. 637–646. ACM, New York (2009)
23. Nicholson, A.J., Noble, B.D.: BreadCrumbs: Forecasting Mobile Connectivity. In: Proceedings of MobiCom 2008, pp. 46–57. ACM, New York (2008)
24. Piorkowski, M., Sarafijanovic-Djukic, N., Grossglauser, M.: CRAWDAD trace set epfl/mobility/cab (v. 2009-02-24) (February 2009), http://crawdad.cs.dartmouth.edu/epfl/mobility/cab
25. Schreiber, T.: Efficient Neighbor Searching in Nonlinear Time Series. International Journal on Bifurcations and Chaos 5, 349–358 (1995)
26. Song, L., Deshpande, U., Kozat, U.C., Kotz, D., Jain, R.: Predictability of WLAN Mobility and its Effects on Bandwidth Provisioning. In: Proceedings of INFOCOM 2006 (April 2006)
27. Song, L., Kotz, D.: Evaluating Location Predictors with Extensive Wi-Fi Mobility Data. In: Proceedings of INFOCOM 2004, pp. 1414–1424 (2004)
28. Zhou, C., Frankowski, D., Ludford, P., Shekhar, S., Terveen, L.: Discovering Personally Meaningful Places: An Interactive Clustering Approach. ACM Trans. Inf. Syst. 25(3), 12 (2007)

Efficient neighbor searching in nonlinear time series analysis

Thomas Schreiber
Department of Theoretical Physics, University of Wuppertal,
D–42097 Wuppertal

July 18, 1996

We want to encourage the use of fast algorithms to find nearest neighbors in k–dimensional space. We review methods which are particularly useful for the study of time series data from chaotic systems. As an example, a simple box–assisted method and possible refinements are described in some detail. The efficiency of the method is compared to the naive approach and to a multidimensional tree for some exemplary data sets.

1 Introduction

Finding nearest neighbors in k–dimensional space is a task encountered in many data processing problems. In the context of time–series analysis *e.g.* it occurs if one is interested in local properties in a reconstructed phase space. Examples are predictions, noise reduction or Lyapunov exponent estimates based on local fits to the dynamics, or the calculation of dimension estimates. Other applications in physics include simulations of molecular dynamics with finite range interactions, where a box–oriented approach is used called "link–cell algorithm."[Fincham & Heyes, 1985, Form *et al.*, 1992]

As long as only small sets (say $n < 1000$ points) are evaluated, neighbors can be found in a straightforward way by computing the $n^2/2$ distances between all pairs of points. However, numerical simulations and to an increasing degree experiments are able to provide much larger amounts of data. With increasing data sets efficient handling becomes more important.

Neighbor searching and related problems of computational geometry have been extensively studied in computing science, with a rich literature covering both theoretical and practical issues. General references include [Sedgewick, 1990, Preparata & Shamos, 1985, Gonnet & Baeza–Yates, 1991, Mehlhorn, 1984]. In particular, the tree–like data structures are studied in [Omohumdro,

1987, Bentley, 1980, Bentley, 1990], and the bucket (or box) based methods in [Noga & Allison, 1985, Devroye, 1986, Asano *et al.*, 1985].

Although considerable expertise is required to find and implement an optimal algorithm, we want to demonstrate in this paper that with relatively little effort a substantial factor in efficiency can be gained. The use of any intelligent algorithm can result in reducing CPU time (or increasing the maximal amount of data which can be handled with reasonable resources) by orders of magnitude, compared to which the differences among these methods and the gain through refinements of an existing algorithm are rather marginal. Thus we give a simple and general algorithm which is worth the effort even for sets of only moderate size.

The box–assisted algorithm given here has been heuristically developed in the context of time series analysis [Grassberger, 1990, Grassberger *et al.*, 1991]. Similar procedures are proposed for this purpose in [Theiler, 1987, Kostelich & Yorke, 1988], while the k–d–tree (Sec. 2) seems to be the most popular approach [Bingham & Kot, 1989, Farmer & Sidorowich, 1988]. In Sec. 3 we describe a very simple version of a box–assisted algorithm for finding all points closer than a given distance ϵ. In Sec. 4 we describe how it can be improved by using *linked lists*. To illustrate the usefulness of this data structure, a very fast sorting method is presented. Furthermore we describe how to modify the basic algorithm in order to find a given number of neighbors rather than a neighborhood of fixed diameter. In Sec. 5 we will discuss some examples of the performance of the algorithms described. For comparison we give results obtained using the k–d–tree algorithm described in [Bingham & Kot, 1989], which represents a similar level of sophistication.

Both the box–assisted and the tree implementation used were chosen rather for simplicity than for optimality. The reader who wants to go beyond this will find some suggestions in Sec. 6.

2 The classical approach: multidimensional trees

How to find nearest neighbors in k–dimensional space? The textbook answer is to use multidimensional trees. From the point of view of computing theory they have the advantage that they can be proven to have an asymptotic number of operations proportional to $n \log n$ for a set of n points, which is the best possible performance for sets of arbitrary distribution.

We will see later that there exist algorithms which are faster (the number of operations is asymptotically proportional to n) provided the set is not too singularly distributed. Nevertheless, since trees are very widely applicable and accordingly very popular, we want to recall the basic idea. The algorithm is described in more detail in [Bingham & Kot, 1989].

Neighbor searching consists of two steps, first a data base is built and then this base is scanned in an efficient way to extract the required points. In this

section we want to use a tree–like structure as a data base. At each branching point, the remaining data set is split into two branches according to the result of a comparison. Say we have n vectors of Cartesian coordinates in k dimensions. Since there is no ordering relation in $k > 1$ dimensions we use one of the k coordinates at each level of the tree to determine into which branch a point falls. We cyclically use all the coordinates. The method is illustrated in Fig. 1. Since we will have n branching points and on average $\log_2 n$ levels, the construction of the tree requires $O(n \log n)$ operations.

Once the tree has been built up, we can now use it to find all points closer than ϵ to a given reference point $\mathbf{x}_0$. Instead of computing the distances to all the other points in the set we can now exclude branches which fall completely outside an ϵ–cube around $\mathbf{x}_0$.

3 Box methods: The basic algorithm

Consider a set of n vectors $\mathbf{x}_i$ in k dimensions, for simplicity rescaled to fall into the unit cube. For each x_i we want to determine all neighbors closer than ϵ,[1] or, strictly speaking, determine the set of indices

$$\mathcal{U}_i(\epsilon) = \{j : \ ||\mathbf{x}_j - \mathbf{x}_i|| < \epsilon\} . \tag{1}$$

This is preferable for technical reasons since we do not want to move the whole vectors around, in particular not if they are obtained as delay coordinates from a scalar time–series.

The idea of box–assisted methods is the following. Divide the phase space into a grid of boxes of side length ϵ. Then each point falls into one of these boxes. All its neighbors closer than ϵ have to lie in either the same box or one of the adjacent boxes. Thus in two dimensions only 9 boxes have to be searched for possible neighbors, *i.e.* for $\epsilon < 1/3$ we have to compute less distances than in the simple approach.

The technical problem is how to put the points into the boxes without wasting memory. How to provide the right amount of memory for each box without knowing beforehand how many points it will contain? The conceptually simplest solution is to obtain this information by first filling a histogram which tells how many points fall into which box. With this information one can divide an array of n elements so that each box is assigned a contiguous section of memory exactly as large as to hold all the points in the box (see Fig. 2). For each box one has to store the position where this section starts. The end of the section is given by the starting position for the next box. Once the indices of all the points are stored in the section corresponding to the appropriate box, scanning a box simply means scanning its section of the array.

We suggest the following implementation.

[1] Throughout this paper we use the sup norm. Neighbors in the Euclidean or the L^1 norms are also neighbors in sup norm.

1. To save memory we use two–dimensional boxes regardless of the dimension k, unless the set itself is of very high dimension. Neighbors in two dimensions are the candidates for neighbors in k dimensions. The two least correlated coordinates promise most information in two dimensions.

2. Provide an array `BOX` of size $n \times n$. If $n\epsilon > 1$ we waste memory. If $n\epsilon < 1$ we have to "wrap around" the set since the boxes do not cover the whole area. Thus we take the coordinates $\{x_j^{(k)} \bmod (n\epsilon)\}$ instead of $\{x_j^{(k)}\}$ to determine into which box a point falls.

3. Provide a linear array `POINTS` with n elements to contain the indices of all points.

4. Use `BOX` to make a histogram of how many points fall into each box.

5. Replace the histogram in `BOX` by an accumulated histogram, going through `BOX` row by row, say from left to right, and sum the number of points to be stored so far. Now each element of `BOX` tells where the section of `POINTS` corresponding to this box *ends*.

6. For every point determine into which box it falls, say `(I,J)`. Store the index of this point at the location on `POINTS` given by the current value of `BOX(I,J)`. Count down `BOX(I,J)` one step.

7. When all points are inserted, the elements of `BOX` tell where the corresponding sections of `POINTS` *start*.

8. In order to scan a box, just scan the corresponding section of `POINTS`. It ends where the section of the next box begins. Each candidate is tested whether it is a neighbor in the desired norm in k dimensions.

The resulting FORTRAN program is given as algorithm 1. It consists of two subroutines. The first one, `BOXIT`, scatters the points into a square array `BOX` and the appropriate sections of array `POINTS`. The second one, `FIND`, finds $\mathcal{U}_i(\epsilon)$ for a given index i. The indices of these points are stored in array `FOUND`.

The method works most efficiently if the expectation value of the number of points per box is $O(1)$. For small box size ϵ this can be achieved by providing roughly as many boxes as there are points. Thus we need $2n$ storage locations in addition to the data. For 2^{16} points or less, the arrays `BOX` and `POINTS` can be 2–byte integers.

4 Linked lists and sorting

The above algorithm can be slightly accelerated by the use of *linked lists* [Knuth, 1973] instead of linear arrays. We can save the time needed for forming and accumulating the histogram, while the storage needed is exactly the same. With

linked lists points can be added to the data base at any time, a useful feature for real–time applications. Thus it is also possible to compress filling of boxes and retrieval of neighbors into one sweep through the data, as was done in [Grassberger, 1990].

A one–directional linked list is a data structure in which each element contains a data item (or a pointer to it) and a pointer to the next element in the list. A pointer to a *null* location indicates the end of the list. To access the list we have to know where its first element is stored. A new element is inserted into the list by redirecting the pointer of the preceding element and letting the new element point to the following one (see Fig. 3). Note that we do not want to move the data items (the phase space vectors $\mathbf{x}_i$) around. We can save the storage for the pointers to these items by exploiting the one–to–one correspondence between list elements and data items: list elements and data items have the same index.

To illustrate the usefulness of this data structure let us for a moment think about the problem of ranking (or sorting) n numbers $x_i \in [0,1]$. To begin with, one could implement the method of *straight insertion*. First the list has only one element, x_1. Now the other numbers are added to the list consecutively. Each new number is either inserted between a smaller and a larger element, added at the end if it is larger than the last element, or added at the beginning if it is smaller than the first element. The number of comparisons required by this procedure is $\propto n^2$.

A box–assisted technique can be used to accelerate the algorithm to an operation count $\propto n$. The idea is to guess from the size of x_i where it has to be inserted. We divide the interval into n boxes and store for each box where the section of the list corresponding to this value starts. Inserting x_i we have to determine into which box it falls and scan only through the corresponding section of the list. For non–singularly distributed points the sections have a mean length of n/m. Sorting n points with the help of $m \propto n$ boxes yields an algorithm requiring $\propto n$ operations. For non–uniformly distributed points, the operation count is $\propto n^2/m^{d_2}$, where d_2 is the correlation dimension of the distribution.

Algorithm 2 shows a FORTRAN subroutine for ranking `NMAX` real numbers stored in the array `X`, and storing their ranks in the array `LIST`. It does this by means of an auxiliary array `BOX` of size `NBOX`+1. After calling the subroutine, `LIST(I)` contains the rank of `X(I)`, which is defined as {1 + the number of `X(J)` less than `X(I)`}. For equal values the time order is taken. It is easy to modify this algorithm so that it gives ordered linked lists as output.

Ranking the 50,000 points in the data set (iii) with `NBOX` = `NMAX`/2 takes 0.48 sec. of CPU time on a SPARCstation 1+ (average value of several runs). The fastest known comparison–based algorithm, quicksort (as implemented in [Press *et al.*, 1988]), takes 1.07 sec, slower by a factor of more than 2. For more evenly distributed data, this factor can increase up to 4.

In the neighbor search problem in k dimensions we use the same idea, only

now the lists starting at each box do not have to be sorted. The lists corresponding to each box all end with a *null* pointer to mark where to stop scanning.

Efficient neighbor searching with a fixed grid of boxes requires the box size to be equal to the radius of the neighborhoods we want to find. However, if one wants to find a given number K of nearest neighbors (neighborhoods of *fixed mass*), the procedure has to be modified. It is quite inefficient to choose a small box size ϵ and then scan more and more boxes until at least K neighbors are found. We rather suggest a slight modification of a scheme applied in [Schreiber & Grassberger, 1991] in the context of noise reduction. It consists of the following steps:

1. Start with a very fine partition of box length ϵ.
2. For all points find the neighbors within a distance less than ϵ.
3. Some of the points will have less than K neighbors with this resolution ϵ. These points are marked for the next sweep.
4. Coarsen the partition by a linear factor $\alpha > 1$, $\epsilon' = \alpha\epsilon$. With $\alpha = 2^{1/d}$, where d is the dimension of the set, one roughly doubles the mass of the neighborhoods. A new data base is created containing all points in boxes of size ϵ'.
5. Now only for the marked points find all the neighbors within a distance less than ϵ'.
6. Coarsen the partition until enough neighbors have been found for all points.

Note that we first satisfy the requirement of K neighbors for points in dense regions. Points in sparse regions are served last. If one needs to find neighborhoods for central points according to a given order, the method fails. In this case the algorithm of the last section can be modified by mapping the linear array `POINTS` to the square array `BOX` via a space filling Peano curve [Simmons, 1963]. Then boxes of different sizes $2^l\epsilon, l = 0, 1, \ldots$ map to contiguous sections of the array. Although this can be implemented efficiently, we prefer the use of trees for this problem.

5 Results and comparison

In this section we will give some typical results for the CPU time required by the methods described above. How long it takes to find nearest neighbors depends on the number of points n, the dimension of the set, the embedding dimension and either the radius ϵ of the neighborhoods or the minimal number of neighbors required. Rather than scanning through a four–dimensional parameter space we

will concentrate on three data sets, (i) a data set consisting of 1,000 iterations of the Ikeda map, (ii) 10,000 iterates of the same map, and (iii) 50,000 sampless taken at one site of a coupled map lattice which is in the state of space–time chaos, representing a high dimensional example.

We embed the first two samples in 4 dimensions and the third in 3, 5 and 10 dimensions. All CPU times were obtained on a SPARCstation 1+ but even their relative magnitudes will be slightly machine dependent.

We implemented a k–d tree as described in [Bingham & Kot, 1989], also using FORTRAN for comparison. If a language supporting pointers and recursive function calls (*e.g.* C) is used, it is about as easy to implement as the algorithm we present. (It will usually not be faster than the FORTRAN implementation though). For possible improvements of this algorithm see Sec. 6.

Even for the short set (i) the box method pays compared to the naive approach below $\epsilon = 1/3$ where both take about 7s. Only for very small $\epsilon < 1/8$ the tree is faster than the naive method, while it is consistently a factor of two slower than the box method. The time to find 50 neighbors is 11s for the naive, 6s for the tree and 4s for the box methods.

The results for the longer ikeda sample (n=10,000) are given in Table 1. Three different values of the fixed diameter ϵ were chosen: a quite small value (1/32 of the total attractor size), a typical value below which scaling could be expected (1/16), and a larger value (1/8). Also with this data set the box method was somewhat faster than the tree.

For the high–dimensional data in set (iii), the naive approach would take several hours of CPU time due to the large number of points. On this set the box method is fastest for fixed size neighborhoods below 1/16 of the linear extension of the phase space, where it takes about 60s to find all neighbors. For larger ϵ the tree wins slightly.

For fixed mass neighborhoods the situation depends on the embedding dimension. While for the three dimensional embedding the two algorithms were about equally fast (*e.g.*, finding 50 neighbors takes 760s), for the high embedding dimensions the tree is clearly faster (by a factor of 1.5–2), even if a grid of boxes in up to 5 dimensions is applied. In our method each box requires a storage location even if it is empty. Thus the dimension of the grid can only be increased under coarsening of the resolution, unless one has a lot of memory to spend. Circumventing this problem is addressed in the following section.

6 Possible improvements

The programs we applied in the above sections are very basic implementations of more general concepts. In this section we give some pointers to the literature which develops this further.

The problem of finding nearest neighbors is related to a whole range of problems of computational geometry [Gonnet & Baeza–Yates, 1991, Mehlhorn,

1984, Bentley, 1990, Asano *et al.*, 1985], problems in which a data base has to be built from a set of points in some geometric space. After that, certain queries have to be supported by this data base. One wants to devise algorithms optimal in terms of the size of storage and number of operations needed for a given task. The asymptotic scaling of the operation count with the number of points n is called the *complexity* (more specific, the *computational complexity*) of an algorithm. In the class of problems under consideration the total operation count comprises the effort to built the data base from the unstructured data and the effort to perform a query.

For the naive method of finding neighbors the effort to build the data base is actually zero since queries are answered using the raw data. The latter takes $O(n)$ operations for each query. With the simple k–d tree we used, the data base (*i.e.* the tree) can be built in $O(n \log n)$ operations on average and a query costs $O(\log n)$ operations. To fill the boxes in our box–algorithm always takes $O(n)$ operations but the query time depends on how many boxes we provided. We can trade storage for time and obtain optimal behavior for $O(n)$ boxes, where we expect to find $O(1)$ points to scan through.

Which algorithm is optimal depends on how many queries will be performed. For *in sample* applications like noise reduction, dimensions etc. the number of queries is equal to the number of data points. This can be different in forecasting. If only a few forecasts are asked for, there is no point in applying a fast algorithm at all. If many forecasts will be asked from a limited data base it will pay to optimize accessability of the data base even if the setup of the data base might be slower.

6.1 Refinements for tree methods

For tree methods several improvements have been proposed [Bentley, 1990, Sproull, 1991]. For all but the simplest implementation of trees it is highly recommended to use a programming language supporting structured data types, pointers and dynamical storage allocation.[2] The simplest k–d tree recursively splits the available space at each node into two halves, the splitting line at level J being parallel to the (J mod k)–th coordinate axis.

Balancing: To get optimal query times, the tree should be *balanced*, *i.e.* branches of the same level should contain about equally many points. However, in more than one dimensions it is hard to construct a set which would lead to a grossly unbalanced tree. This is different from the situation in one dimension, where a simple sorting algorithm based on a binary tree could perform very badly on presorted data.

[2] On some systems the last feature is formally supported but very slow. One can avoid its use as long as the number of nodes etc. is predictable from the number of data points, which is the case for all algorithms mentioned here.

Bucketing: A more relevant improvement is obtained by limiting the spatial resolution of the tree. Thus a *leaf* of the tree no more corresponds to a single data point but to a whole *bucket* of points close to the last node. This in itself reduces the query time to $O(\log m)$ where m is the number of buckets. A further improvement is gained by providing a pointer to the parent node for each node within the tree structure. The thereby possible *bottom up* queries take only $O(1)$ operations.

Principal cuts: While cuts parallel to the coordinate axes are natural if neighborhoods are defined in the sup norm, queries in the euclidean norm can be accelerated by cuts along principal axes of the distribution of points. Of course, this requires more work in constructing the data base. See [Sproull, 1991] for details.

6.2 Refinements for box methods

Clever high–dimensional meshes: As we have seen working on data set D, the algorithm given in this paper works best on not too high–dimensional problems. The reason is that in order to cover a high–dimensional space with $O(n)$ boxes of size ϵ we have to wrap around the set several times. Eventually only a small fraction of the points we find in a box will be real neighbors, most will be wrapped images of far away points. The box–assisted method proposed by Theiler [Theiler, 1987] circumvents this problem. An arbitrarily fine grid of boxes can be provided by storing only a table of the non–empty boxes. Thus to scan a neighboring box one has to look up the location of this box in the table. If the table has been organized by a *lexicographical sort*, queries can be done with $O(\log n)$ operations. This and the higher ($O(n \log n)$) cost for constructing the data base pays mainly in higher dimensions.

Adaptive buckets: The sorting algorithm we gave performs best on fairly uniform sets. For singularly distributed sets the use of multiple keys or *adaptive buckets* will be necessary (see [Noga & Allison, 1985]).

6.3 Alternatives for special problems

The algorithms described so far can be used as general purpose routines. However, one often has some additional knowledge about the data or one has to deal with a simpler task. For some cases the algorithms given above can be accelerated, one might even think of implementing a special purpose algorithm for the case at hand.

For some problems one does not really need the set of neighbors itself but only the number of neighbors within a given maximal distance. With the algorithms described so far this can accelerate queries if we can access the number

of points in each box or, using trees, store at each node the size of the subtree starting at this node.

One frequently deals with data of low significant resolution, either because the data is given in coarsely discretized form or only part of the given information is relevant due to noise. In this case it can be possible to form a histogram at the resolution of the data and use this histogram to count neighbors. If the number of distinguishable states is too high to provide bins for all of them, one can again store only the nonempty bins.

An efficient algorithm for finite resolution data can be obtained as a special case of the box–algorithm by Theiler [Theiler, 1987] mentioned above. If one box is provided for each distinguishable state, both the setup of the data base and the retrieval of neighbors can be done very fast.

On a wide range of problems even the simple tree– and box–assisted methods are both found to be quite efficient, the differences being rather marginal. In some cases (finding fixed mass neighborhoods in sets of high dimension) tree methods are faster, in other cases (sets of lower dimensions, fixed radius neighborhoods) box methods win. The differences are in no case strong enough to favour one of the methods in general.

However, we feel that the barrier of implementation of any fast neighbor search method is quite low with the simple algorithm given here, which we found much easier to implement (and to modify) than even the simple multidimensional tree [Bingham & Kot, 1989] we used for comparison.

Acknowledgements

I am grateful to Holger Kantz and Peter Grassberger who closely accompanied the work presented in this paper. Neil Gershenfeld drew my attention to alternative approaches for finite resolution data. The author receives a grant within the framework of the SCIENCE programme of the Commission of the European Communities under contract no B/SC1*-900557.

References

[Asano *et al.*, 1985] T. Asano, M. Edahiro, H. Imai, M. Iri and K. Murota, *Practical Use of Bucketing Techniques in Computational Geometry*, Computational Geometry, G. T. Toussaint, ed., Elsevier (1985).

[Bentley, 1980] J. L. Bentley, *Multidimensional Divide–and–Conquer*, Communications of the ACM, **23**, 214 (1980).

[Bentley, 1990] J. L. Bentley, *K-d Trees for Semidynamic Point Sets*, Sixth Annual ACM Symposium on Computational Geometry, 91, San Francisco, (1990).

[Bingham & Kot, 1989] S. Bingham and M. Kot, *Multidimensional trees, range searching, and a correlation dimension algorithm of reduced complexity* Phys. Lett. **A 140**, 327 (1989).

[Devroye, 1986] L. Devroye, *Lecture Notes on Bucket Algorithms*, Progress in Computer Science no. 6, Birkhäuser, Boston, (1986).

[Farmer & Sidorowich, 1988] J.D. Farmer and J. Sidorowich, *Exploiting Chaos to Predict the Future and Reduce Noise*, in "Evolution, Learning and Cognition", Y.C. Lee, ed., World Scientific, (1988).

[Fincham & Heyes, 1985] D. Fincham and D. M. Heyes, in "Dynamical Process in Condensed Matter – Advances in Chemical Physics," vol. LXIII, M. W. Evans, ed., Wiley, New York, (1985).

[Form *et al.*, 1992] W. Form, N. Ito, and G. A. Kohring, *Vectorized and parallelized algorithms for multi–million particle MD–simulation*, to appear in Int. Jour. Mod. Phys. C, (1992).

[Gonnet & Baeza–Yates, 1991] G. H. Gonnet and R. Baeza–Yates, *Handbook of Algorithms and Data Structures, in Pascal and C*, Addison–Wesley, Wokingham (1991).

[Grassberger, 1990] P. Grassberger, *An optimized box–assisted algorithm for fractal dimensions*, Phys. Lett. **A 148**, 63 (1990).

[Grassberger *et al.*, 1991] P. Grassberger, T. Schreiber and C. Schaffrath, *Nonlinear time sequence analysis*, Int. J.Bifurcation and Chaos **1**, 521 (1991).

[Knuth, 1973] D. E. Knuth, *The art of computer programming*, Vol 1. *Fundamental algorithms* and Vol 3., *Sorting and Searching*, Addison–Wesley, Reading, (1973).

[Kostelich & Yorke, 1988] E. J. Kostelich and J. A. Yorke, *Noise reduction in dynamical systems*, Phys. Rev. **A 38**, 1649 (1988).

[Mehlhorn, 1984] K. Mehlhorn, *Data Structures and Algorithms 3: Multi–dimensional Searching and Computational Geometry*, Springer (1984).

[Noga & Allison, 1985] M. T. Noga and D. C. S. Allison, *Sorting in Linear Expected Time*, Bit 25 (1985) 451–465.

[Omohumdro, 1987] S.M. Omohumdro, *Efficient Algorithms with Neural Network Behavior*, Complex Syst. **1**, 273 (1987).

[Preparata & Shamos, 1985] F. P. Preparata and M. I. Shamos, *Computational Geometry, an Introduction*, Springer, New York (1985).

[Press *et al.*, 1988] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, *Numerical Recipes*, Cambridge Univ. Press, (1988).

[Sedgewick, 1990] R. Sedgewick, *Algorithms in C*, Addison–Wesley, Reading, (1990).

[Schreiber & Grassberger, 1991] T. Schreiber and P. Grassberger, *A simple noise–reduction method for real data*, Phys. Lett. **A 160**, 411 (1991).

[Simmons, 1963] G. F. Simmons, *Introduction to Topology and Modern Analysis*, McGraw–Hill, Tokyo, (1963).

[Sproull, 1991] R. F. Sproull, *Refinements to nearest–neighbor searching in k-d trees*, Algorithmica **6** (1991) 579.

[Theiler, 1987] J. Theiler, *Efficient algorithm for estimating the correlation dimension from a set of discrete points*, Phys. Rev. **A 36**, 4456 (1987).

Figure captions

1. How the plane is divided into branches of a tree.
2. How points within each box are stored in sections of an array.
3. How a new element is inserted into an existing list.

Table caption

1. CPU times (seconds) for data set (ii), n=10,000

Fig. 1

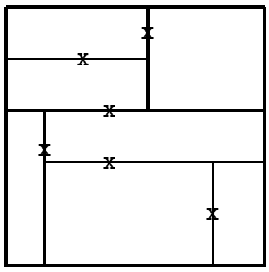


Fig. 2

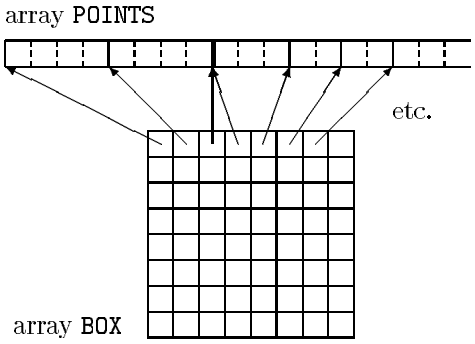
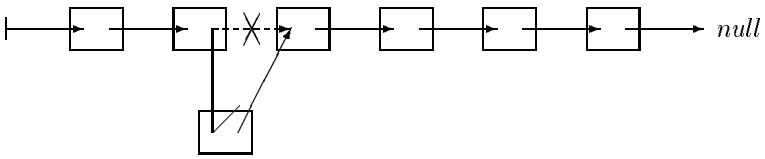


Fig. 3



task		box	tree	naive
fixed radius				489
	$\epsilon = 1/32$	5.6	9.8	
	$\epsilon = 1/16$	19.9	21.0	
	$\epsilon = 1/8$	72.2	75.9	
fixed mass	$m = 50$	45	64	675

Algorithm 1

```
      SUBROUTINE BOXIT(NMAX,X,K,EPS,BOX,POINTS)
```

This routine sets up the box data base for neighbor search in K+1 dimensional delay coordinates. Give a REAL array X containing NMAX points and the box size EPS. The data base is stored in arrays BOX and POINTS. The bitwise .AND.IM is used to to take an integer modulo IM+1

```
      PARAMETER(IM=127)   ! IM must be 2**l-1 for integer l
      REAL X(NMAX)
      INTEGER BOX(0:IM,0:IM), POINTS(NMAX)

      EPSINV=1./EPS

      DO 1000 I1=0,IM
         DO 1000 I1=0,IM
1000        BOX(I1,I2)=0

      DO 2000 N=1,NMAX-K   ! make histogram
         I1=INT(X(N)*EPSINV).AND.IM
         I2=INT(X(N+K)*EPSINV).AND.IM
2000     BOX(I1,I2)=BOX(I1,I2)+1

      DO 3000 I1=0,IM       ! accumulate it
         DO 3100 I2=1,IM
3100        BOX(I1,I2)=BOX(I1,I2)+BOX(I1,I2-1)
3000     IF(I1.LT.IM) BOX(I1+1,0)=BOX(I1+1,0)+BOX(I1,IM)

      DO 4000 N=1,NMAX-K   ! fill boxes
         I1=INT(X(N)*EPSINV).AND.IM
         I2=INT(X(N+K)*EPSINV).AND.IM
         POINTS(BOX(I1,I2))=N
4000     BOX(I1,I2)=BOX(I1,I2)-1
      END
```

```
      SUBROUTINE FIND(NMAX,X,N,K,EPS,BOX,POINTS,FOUND,NFOUND)
```

Given the data base set up by BOXIT, this routine finds all neighbors closer than EPS for the point with index N. The indices of the NFOUND neighbors found are stored in INTEGER array FOUND

```
      PARAMETER(IM=127)    ! same value as in BOXIT
      REAL X(NMAX)
      INTEGER BOX(0:IM,0:IM), POINTS(NMAX), FOUND(NMAX)

      EPSINV=1./EPS
      NFOUND=0
      I1=INT(X(N)*EPSINV).AND.IM
      I2=INT(X(N+K)*EPSINV).AND.IM

      DO 1000 J2=I2-1,I2+1    ! search neighboring boxes
         DO 1000 J1=I1-1,I1+1
            L1=J1.AND.IM
            L2=J2.AND.IM
            IF(L2.LT.IM) THEN     ! determine end of section
               IEND=BOX(L1,L2+1)
            ELSE IF(L1.LT.IM) THEN
               IEND=BOX(L1+1,0)
            ELSE
               IEND=NMAX-K
            ENDIF

            DO 1100 ISCAN=BOX(L1,L2)+1,IEND  ! scan box
               IPOINT=POINTS(ISCAN)
               DO 1110 M=0,K                 ! sup norm in k+1 D
1110              IF(ABS(X(N+M)-X(IPOINT+M)).GE.EPS) GOTO 1100
               NFOUND=NFOUND+1               ! it's a neighbor
               FOUND(NFOUND)=IPOINT
1100           CONTINUE
1000        CONTINUE
      END
```

Algorithm 2

```
      SUBROUTINE RANK(XMAX,NMAX,X,LIST)
```

Rank NMAX points given in X in the inteval [0,XMAX]. Ranks are stored in array LIST. It is assumed that in a composed logical expression containing .OR. or .AND. the first part is evaluated first. Depending on the result, the second part should not be evaluated. For other compilers the IF statements containing .OR. or .AND. have to be split into two statements.

```
      PARAMETER(NBOX=100000)
      REAL X(NMAX)
      INTEGER LIST(NMAX),BOX(0:NBOX)

      NL=MIN(NBOX,NMAX/2)
      SC=(NL-1)/XMAX
      DO 1000 I=0,NL
1000     BOX(I)=0

      DO 2000 N=1,NMAX
         XN=X(N)
         I=INT(XN*SC)
         IP=BOX(I)
         IF ((IP.EQ.0).OR.(XN.LE.X(IP))) THEN
            BOX(I)=N
         ELSE
  2         IPP=IP
            IP=LIST(IP)
            IF ((IP.GT.0).AND.(XN.GT.X(IP))) GOTO 2
            LIST(IPP)=N
         ENDIF
2000     LIST(N)=IP

      N=0
      DO 3000 I=0,NL
         IP=BOX(I)
  3      IF (IP.GT.0) THEN
            N=N+1
            IPP=IP
            IP=LIST(IP)
            LIST(IPP)=N
            GOTO 3
         ENDIF
3000  CONTINUE
      END
```