2018 IEEE 30th International Conference on Tools with Artificial Intelligence

A Rich-Dictionary Markov Predictor for Vehicular Trajectory Forecasting

Dimitrios Papakostas, Dimitrios Katsaros
Department of Electrical & Computer Engineering, University of Thessaly, Greece

***Abstract*—Next-location prediction in a VANET system, where each vehicle acts as a network node, is of great importance in intelligent transport systems (ITS) as this property could have a direct and positive effect on network connectivity, traffic management and hence, improve overall ITS safety. In the last few years, the widespread use of GPS navigation systems and wireless communication technology-enabled vehicles has resulted in huge volumes of trajectory data. The task of utilizing this data employing pattern-matching techniques for next-location prediction in an efficient and accurate manner is an ongoing research problem. This paper presents the Rich-Dictionary Markov Predictor (RDM), a protocol for producing online these forecasts by using a pattern matching technique. RDM is fast, accurate and fully parameterized presenting different trade-offs as regards efficiency versus prediction accuracy. We evaluated the effectiveness of RDM via simulation and the results attest that it achieves on the average more than 35% better prediction accuracy and competitive to faster prediction times than other model independent and highly accurate prediction algorithms.**

I. Introduction

Vehicular ad hoc networks (VANETs), are spontaneous, flexible wireless networks that are able to support the associated applications in dynamic, multi-hop topologies. However, the relatively high speed of the moving vehicles degrades link quality, causes fast fading, short connectivity and high frequency hand-offs. In general, such mobility related problems are addressed by appropriate broadcasting techniques, by smart routing, by clustering protocols [4]. Nevertheless, as Kolios et al. explain in [12], if mechanical relaying; i.e., "store-and-carry" is allowed for the vehicles, then "*a plethora of different, novel resource-utilization schemes can be explored to increase network performance*". Evidently, one of the key components for achieving relaying is the ability to predict vehicles trajectories accurately.

It has been observed [17] that in practice, weekdays and weekends usually exhibit significantly different traffic conditions, whilst at the same time having similar congested and congestion free traffic patterns. Based on that observation it is straightforward to conclude that every vehicle does not follow the same path every time they leave their base, e.g., house. These different moving patterns relate to time of day such as driving to work in the morning and hobbies in the evening and also the entry point in the road network, which probably means a different final location. In such a realistic situation, where the actual path of each vehicle is not known in advance and most of the vehicles enter some areas of the city, e.g., city centre, on the same time period, the existence of a fast and accurate prediction mechanism is beneficial to:

- Routing protocols; i.e., the selection of the next hop is a necessary ingredient of store-carry-forward algorithms [11], and also for geocasting protocols [9].
- Traffic management: traffic management applications focus on improving the vehicle traffic flow and traffic assistance. Possible converging vehicle paths might provide drivers useful information so that they can make the best decisions in terms of their route, such as avoiding congested areas.
- Connectivity robustness: user applications which provide value added services like the Internet, and p2p applications, would exploit proposed (predicted) paths that would guarantee an acceptable Quality of Service (QoS) for these applications.
- Safety: drivers are proactively informed about possible conflicting paths between neighboring /approaching cars.

In this article, we propose a new next-location prediction scheme, namely the Rich Dictionary Markov (RDM) predictor. The article makes the following contributions:

- It exploits the resource-rich environment (battery, computing power and storage) of a vehicle to build a rich summary of its roaming history that subsequently is used to provide more accurate predictions.
- It uses data structures that are constructed in a purely distributed fashion.
- It develops a new forecasting model that is a combination of two prediction mechanisms.
- The proposed algorithm is fully parameterized, presenting different trade-offs in efficiency vs. prediction accuracy.
- It provides a comparison of the proposed method against several, model independent and highly accurate prediction algorithms, and the results show that RDM achieves on the average:
  - More that 35% better prediction accuracy than the second best-performing algorithm.
  - Competitive to faster prediction times than its competitors.

The rest of this article is organized as follows: First, we provide the technical insight of the prediction algorithms and explain how the route prediction problem can be modelled as a discrete symbol prediction problem (§ II). Then, we briefly survey related works (§ III), we unveil the unique characteristics of RDM (§ IV), we present its prediction mechanism (§ V) and evaluate its performance (§ VI). Finally, we conclude the article (§ VII).

2375-0197/18/$31.00 ©2018 IEEE
DOI 10.1109/ICTAI.2018.00091

II. Technical insight of prediction algorithms

Consider a sequence of position updates $\{x_1, x_2, \ldots x_i\}$ being generated by a vehicle, represented by the stochastic process $X = \{x_i\}$. A predictor will have to predict what the next position x_{i+1} is going to be on the basis of the observed history while minimizing the prediction errors over the course of an entire sequence. Authors in [5] proved the existence of universal predictors that could optimally predict the next item in any deterministic sequence and argued that an optimal predictor must belong to the set of all possible finite state machines. They also showed that universal FS predictors achieve the best possible sequential prediction that any FSM can make and that Markov predictors, a subclass of FS predictors, perform as well as any finite state machine. Moreover, they highlighted that a Markov predictor whose order grows with the number of symbols in the input sequence attains optimal predictability faster than a predictor with a fixed Markov order. Finally, they proved that Markov predictors based on the $LZ78$ incremental parsing algorithm attain optimal predictability because they achieve to changing the Markov order rapidly enough to reach a high order of Markov predictability and slowly enough to gather sufficient information at each order of the model to reflect the model's true nature.

A. Markov predictors

To provide the formal definition of the prediction model we use in our work we follow the work in [10]; a trajectory a_i of a vehicle i is a finite sequence of symbols a_i^j drawn from an alphabet Σ (where $a_i^j \in \Sigma, \forall\, i, j$), with each symbol a_i^j standing for a road-segmID. A predictor accumulates sequences of the type $a_i = a_i^1, a_i^2, \ldots a_i^{n_i}$, where n_i denotes the number of symbols constituting a_i. Without loss of generality, we can assume that all the knowledge of the predictor consists of a single sequence $a = a^1, a^2, \ldots a^{n_i}$. Based on a, the predictor's goal is to construct a model that assigns probabilities for any future outcome given some past. As it is stated in [10], this formulation implies a stochastic process $(X_t)_{t \in N}$ where at any given time instance t (meaning that t symbols $x_t, x_{t-1}, \ldots x_1$ have appeared, in reverse order), we need to calculate the conditional probability :

$$\bar{P}[X_{t+1} = x_{t+1} | X_t = x_t, X_{t-1} = x_{t-1}, \ldots], \qquad (1)$$

where $x_i \in \Sigma, \forall\ x_{t+1} \in\ \Sigma$.

Predictors that use this kind of prediction model are termed higher-order Markov predictors and the history $x_t, x_{t-1}, \ldots$ used in the above definition is called the context of the predictor. These predictors maintain a set of relative frequency counts for the symbols seen at different contexts in the sequence, thereby extracting the sequence's inherent pattern. They then use these counts to generate a posterior probability distribution for predicting the next symbol to come.

B. The vehicle route prediction problem as a discrete symbol prediction problem

In our work we model the road network as a directed graph G = (V, E), where the set V represents the road segments and the set E represents the directed connectivity links between pairs of road segments. Each segment has its own identification number, namely, road-segmID (Fig. 1).

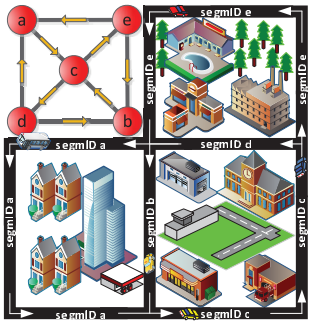

Fig. 1. A sample road network and its corresponding graph; the arrows in the road network and graph depict permitted direction of movement. SegmIDs are represented by nodes.

The segmentation of a road network into segments is a process similar in principle to that of designing location areas for a cellular network [3] [15]. Vehicles are assumed to perform random walks over this network and is also taken that each vehicle is aware of the road-segmID that is moving on, by the use of technologies such as GPS. These position updates, not only are they used to build the moving history of each vehicle but in the form of a discrete symbol sequence they are shared between communicating vehicles in order to support their prediction process.

III. Related work

The backbone of the LZ family of prediction algorithms is LZ78 [19]. LZ78 performs incremental parsing of an input string "$x_1, x_2 \ldots x_i$" into *c(i)* substrings (i.e., phrases) "$w_1, w_2 \ldots w_{c(i)}$", such that $\forall j > 0$, the prefix of the substring w_j (i.e., all but the last character of w_j) is equal to some w_i for $1 < i < j$. Because of this prefix property, parsed substrings and their relative frequency counts can be maintained efficiently in a multiway tree structure, namely, a digital tree or prefix tree (commonly known as trie). An LZ78 parsing of the string "aaababbbbbaabccddcbaaaa" yields the phrases: "a", "aa", "b", "ab", "bb", "bba", "abc", "c", "d", "dc", "ba", "aaa". LZ78 maintains statistics for all contexts seen, for example, the context "a" occurs five times (at the beginning of the phrases "a", "aa", "ab", "abc", "aaa"), the context "bb" is seen two times ("bb", "bba"), etc.

LZ78 has three main drawbacks:

- In any LZ78 parsing of an input string, all the information crossing phrase boundaries is lost. In our example string, the fourth symbol "b" and fifth and sixth symbols "ab" form separate phrases; had they not been split, LZ78 would have found the phrase "bab", thereby creating a larger context for prediction.

- Phrases contained within substrings are also lost;
- The prediction performance of LZ78, is not good for short sequences [5] [10] [18].

LeZi Update (LZU) [2] makes the same parsing of LZ78 algorithm, but instead of adding just the substrings resulting from this parsing, it adds also all the suffixes of each substring to the LZU trie. Therefore, phrases within substrings are taken into account. In our example string the phrase "bc" is added in the LZU dictionary, which is a suffix of the phrase "abc".

The algorithm proposed by Gopalratnam [6], namely Active LeZi (ALZ) is intended to consider the phrases among consecutive parsed substrings, thus solving the remaining problem of LZ78 algorithm and converging faster to optimal predictability. In order to achieve this, ALZ incorporates a window of variable length, which is determined on the fly, without any extra computational overhead, by the longest phrase parsed by LZ78 algorithm at each step. Once the length of the window is updated and a new symbol is added to it, all the suffixes of the window are added to the trie. The detailed performance evaluation of the major Markov predictors in [10] highlighted their shortcomings and suggested routes for their improvement.

Authors in [13] employ the ALZ algorithm to construct frequency trees of a fixed depth k^{max} and perform a Modified Prediction by Partial Match (MPPM) technique in order to obtain a prediction performance that outperforms the best single model predictor. Since the prediction process is online, MPPM utilizes the true data to adaptively weigh the prediction performance of each different order Markov model. The weights are updated at every step and the best performing Markov model is given the highest weight. Thus the problem of finding the best model order for a given sequence length is also implicitly solved by applying the above technique.

Trajectory prediction in VANETS has attracted a significant amount of research recently [16], in order to cope with increased safety issues that arise from the development of autonomous driving technologies [1]. Some recent works deal with lane change prediction [7] [8]; others perform whole trajectory matching with the aim of predicting far-in-the-future positions of a mobile [20], [21]. On the other hand, RDM is a fast, online, next-site prediction model based on analyzing local movement patterns from the recent past.

IV. The Rich Dictionary Markov (RDM) predictor

The RDM predictor is designed for the VANET environment, thus assuming significant energy resources, strong computing power and large storage capacity, by following the suggestions in [10]. Therefore, we enriched the RDM's dictionary, we expanded its trie and intentionally developed a more computation-hungry prediction method.

A. Rich dictionary construction

What differentiates RDM from both ALZ and MPPM is that while they all parse both the input sequence and the phrase that resides in the sliding window, only RDM attaches all these phrases to the dictionary as part of the protocol. The net effect of this procedure is that the algorithms develop completely different dictionaries, tries, sliding window sizes, which altogether affect the prediction accuracy. Algorithm 1 presents the pseudocode for parsing and processing the input sequence in RDM.

Algorithm 1: RDM

precondition : An input sequence
postcondition: An updated dictionary of parsed phrases
remarks : $dictionary$ = stores the parsed phrases, $window$ = a variable length window of previously seen symbols, Max_LZ_Length : the length of the longest parsed phrase, w = a continuously updated phrase that drives the dictionary construction.
initialize : dictionary = null; window = null; Max_LZ_Length = 0;

```
Loop
    Wait for next symbol v;
    if w.v in dictionary then
        w = w.v;
    else
        add w.v in dictionary;
        update Max_LZ_Length if neccessary;
        w = null;
    end
    add v to window;
    if length(window) > Max_LZ_Length then
        delete window[0];
    end
    Update dictionary with all possible prefixes within
      window that include v;
Forever
```

Proposition 1: When the length of the longest phrase parsed by ALZ, RDM and MPPM is less than k^{max} MPPM constructs frequency trees that grow faster.

Proof 1: While ALZ and RDM incorporate a window of variable length, MPPM uses a window of fixed length k^{max}. This modification on the one hand enables MPPM to control the growth of its associated trie (max trie depth = k^{max}) on the other hand allows the phrase that resides in the sliding window (and drives the associated trie construction) to continuously increase its length until it is k^{max}. Therefore, MPPM continuously adds larger phrases in its trie while RDM and ALZ only when a new phrase is entered in each algorithms dictionary. ⊣

Fig. 2 illustrates the tries that the competing algorithms build (k^{max} = 5 for MPPM) by superimposing them into a single trie for the input sequence "aaababbbbbaabccddcbaaaa". We see that MPPM builds the largest trie, however, this is temporal and will stop from happening when the length of the longest phrase parsed by ALZ and RDM becomes greater than k^{max}. In practice RDM's resultant dictionary is richer and the associated trie larger both in span and depth.

Proposition 2: The trie developed by ALZ is strictly contained within that created by RDM.

Proof 2: RDM adds into the dictionary the parsed phrases of the sliding window and therefore, increases the size of the

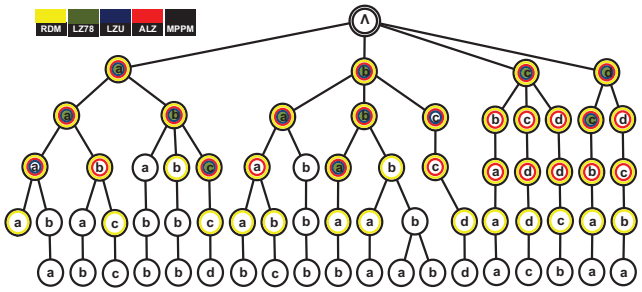

Fig. 2. The trie formed by RDM (Yellow) / LZ78 (Green) / LZU (Blue) / ALZ(Red) / MPPM (Black) after parsing the symbol sequence "aaababbbbbaabccddcbaaaa".

sliding window earlier than ALZ (more precisely it updates earlier the Max_LZ_Length parameter). This larger window, whilst incorporates at any time ALZ's equivalent window, produces more (trie span expansion) and longer phrases (trie depth expansion). ⊣

B. RDM complexity analysis

RDM consumes space and time for data processing. The following analysis discusses the worst-case conditions for the prediction process.

Proposition 3: The space complexity of RDM is $O(n^{\frac{3}{2}})$.

Proof 3: At each step, RDM parses the symbol string within the sliding window and either updates the cardinality of an existing node or adds a new node to the trie. The worst possible case arises when RDM increases the maximum LZ phrase length and the parsed substrings add new nodes to the trie. We represent the sequence of the parsed substrings as $\hat{g} = x_1, x_1x_2, \ldots, x_1x_2 \ldots x_k, where\ |\hat{g}| = n = \frac{k(k+1)}{2}$. A sequence of this form can be represented by an order-k Markov model which stays of order-k through the next k symbols. In the worst case, each parsed substring adds a new node to the trie, so at order k, the trie gains k^2 nodes before the model transitions to order k + 1. Therefore, the number of nodes generated in the trie by the time the model attains order k is $O(k^3) = O(n^{\frac{3}{2}})$, because k = $O(\sqrt{n})$. ⊣

Proposition 4: The time complexity of RDM is $O(n^{\frac{3}{2}})$.

Proof 4: The worst case in terms of runtime arises when RDM parses the worst sequence $\hat{g}$ because it will prompt the most updates. Creating or updating a node in the trie requires finding the appropriate child of a given node along the path that phrase traced in the trie. RDM can access a given node's child in constant time. Therefore, it can find a node in time linear in the depth of the trie. When the worst-case sequence $\hat{g}$ is the case where $|\hat{g}| = n$ and order k = $O(\sqrt{n})$ there must be an update for every order up to k before the model transitions to order k + 1. Consequently, by the time the model attains order k and the number of nodes generated in RDM's trie is $O(n^{\frac{3}{2}})$, the runtime is also $O(n^{\frac{3}{2}})$. ⊣

V. RDM PREDICTION MECHANISM

RDM employs two prediction mechanisms that are used simultaneously and are complementary to each other. The first is purely probabilistic and is similar to that in [2], while the second is deterministic, being based on trie traverse in order to make predictions. The first mechanism considers different order Markov models and employs a blending strategy known as exclusion [2] to build a probability distribution for each state (symbol) occurring in the input string. When used it predicts the state with the highest probability value as the most likely action. We explain how the first mechanism works by referring to Fig. 3, which represents the trie constructed by RDM for the sequence "aaababbbbbaabccddcbaaaa".

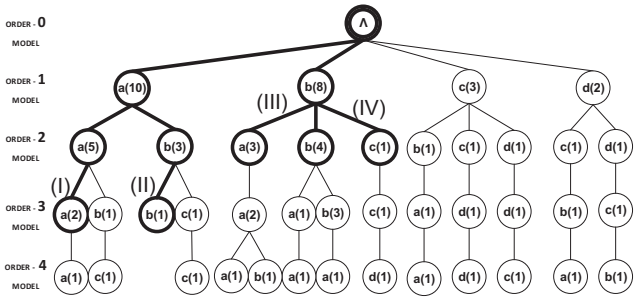

Fig. 3. Different cases (I, II, III, IV) w.r.t the symbol prediction process. The numbers adjacent to each node represent its cardinality in the trie

The window maintained by the predictor represents the set of contexts RDM uses to compute the probability of the next symbol. In our example, the last phrase "aaaa" (which is also the current RDM window) is used. Within this phrase, the contexts that can be used are all suffixes within the phrase, except for the window itself (i.e., "aaa", "aa", "a" and the null context). Suppose we want to calculate the probability that the next symbol is an "a". We see that an "a" occurs one out of two times that the context "aaa" appears. Therefore, $P(a|aaa)$ = $\frac{1}{2}$. Then we fall back to the order-2 context (i.e., the next lower order model) with probability $P_{esc(2)} = \frac{1}{2}$; the so-called escape factor of the order-2 model, which corresponds to the probability that the outcome is null. At this level, we see that it occurs one "a", one "aa" and one "bc" out of five times that the context "aa" appears, therefore $P(a|aa) = \frac{1}{5}$. Then the algorithm falls back (escapes) to the order-1 model with probability $P_{esc(1)} = \frac{(5-(2+1))}{5} = \frac{2}{5}$.

By using the same technique at the order-1 context, we see that an "a" occurs two out of ten times that the "a" appears (the rest belonging to bigger phrases, e.g., "aa", "ab", "aba" etc.) and therefore we have $P(a|a) = \frac{2}{10}$. Then, for the last time, the algorithm falls back (escapes) to the order-0 model with probability $P_{esc(0)} = \frac{2}{10}$ or $\frac{1}{5}$. At that level, we see an "a" two times out of the 23 symbols seen so far, and therefore we predict "a" with probability $P(a|\bigwedge) = \frac{2}{23}$ in the null context. Therefore, in our example, the blended probability of seeing an "a" as the next symbol is:

$$\frac{1}{2} + \frac{1}{2}\{\frac{1}{5} + \frac{2}{5}[\frac{2}{10} + \frac{2}{10}(\frac{2}{23})]\}. \quad (2)$$

The second prediction mechanism employs a purely deterministic approach to make predictions. It uses the continuously

updated LZ phrase "w" that drives the RDM dictionary construction (see the RDMs pseudo-code) to traverse the trie and pinpoint to the current state of the predictor. Note that at each state we record the prediction hits of the predictor regarding that state in the past. RDM exploits the information (history) given from the higher order model and predicts according to the following rules (see Fig. 3):

- Case (I): Single branch at the higher order model → Predict the context given by the higher order model. For example, when w = "aaa", the predictor exploits the information given from the 4th order model and predicts "a" as the next symbol.
- Case (II): No higher order model → Use the probabilistic prediction mechanism and predict the context with the largest probability value, e.g., when w = "abb".

When more than one branches exist under the current state of the predictor, RDM calculates the *Kendall Tau Rank Distance (KTRD)* between the ranking lists of the cardinalities and the prediction hits between the states of the higher order model and uses it as a yardstick for the prediction of the next state. In order to present how RDM works in such cases we will refer in Fig. 3 to the case where w = "b", and we will assume that the prediction hits regarding the states "a", "b" and "c" of the 2nd order model are 2, 2 and 1, respectively. Thus, when w = "b", RDM ranks the states of the higher order model with regards to their cardinality and prediction hits as follows:

State	a	b	c
Rank by Cardinality	2	1	3
Rank by Prediction Hit	1	1	3

Then, RDM pairs each state with every other state and counts the number of times the values in the two lists are not in the same order:

Pair	Cardinality	Prediction Hit	Count
(a,b)	$2 > 1$	$1 = 1$	X
(a,c)	$2 < 3$	$1 < 3$	
(b,c)	$1 < 3$	$1 < 3$	

The calculation of *KTRD* is a simple process which is based on a merge sort algorithm and requires time $O(n \log n)$. If n is the list size, the normalized *KTRD* is :

$$KTRD = \frac{Discordant\ pairs}{n * (n-1)/2} = \frac{1}{3 * (3-1)/2} = 0.33. \quad (3)$$

Then RDM uses the first pair of branches it finds (from left to right) at the higher order model and calculates the difference between their cardinalities. Then, the absolute value of the result is divided by the cardinality of the current status of the predictor (we term it here $RESULT$) and is compared with the *KTRD*. Note that when the higher order model consists of more than two states then Case III and/or Case IV scenarios continue to be executed until all states have been examined:

- Case (III): $RESULT \leq KTRD \rightarrow$ Use the probabilistic prediction mechanism and predict the context with the largest probability value. For example, when w = "b", the first pair of states we examine in the order-2 model is "a", "b". Thus, we have $RESULT = \frac{|a(3)-b(4)|}{b(8)} = \frac{|3-4|}{8} = 0.125$. Since $RESULT < KTRD$ the state with the largest probability value between the two is predicted.
- Case (IV): $RESULT > KTRD \rightarrow$ Predict the context with the larger cardinality. For example, when w = "b", if the pair of states under consideration is "b" and "c", then we have $RESULT = \frac{|b(4)-c(1)|}{b(8)} = \frac{|4-1|}{8} = 0.375$. Since $RESULT > KTRD$ it is predicted the state "b" because it has the largest cardinality.

Practically, the use of *KTRD* enables RDM to speed up the prediction process; it predicts the state with the larger relative cardinality among the states of the higher order model as long as the respective number of discordant pairs between the ranking lists of the cardinalities and the prediction hits remains small.

VI. RDM PERFORMANCE EVALUATION

We perform a simulation-based evaluation of the performance of RDM. To this end, we designed experiments with vehicle itineraries that are segmented and are represented by datasets with discrete symbol sequences that vary with respect to the number and repeatability of patterns within them.

A. Simulation setting

The next paragraphs describe the competing algorithms, the datasets, and the performance measures used.

1) Methods compared: We use the LZ78 [19] compression algorithm as the baseline algorithm for our comparisons due to its simplicity. We implement and evaluate also LeZi-Update [2], Active LeZi [6] and MPPM [5] since the superiority of Markovian predictors over other techniques has been explained in [10].

2) Datasets used: Since there are no publicly available datasets containing real vehicle movements over segmented city roads, we created five datasets in order to evaluate the performance of RDM. Four simulate vehicle itineraries over a road graph with and without noise, and the fifth is a realistic dataset produced using SUMO an open traffic simulation suite.

The first dataset, named s4n0, consists of an alphabet of four different symbols (s4) and contains no noise (n0). It has perfect regularity in terms of the vehicle patterns and a small alphabet (few road-segmIDs) that creates conditions for all the algorithms to have good prediction performance. In the second dataset, named s4n20, we deliberately disrupted the patterns to a percentage of 20%. The third dataset was created by using 20 symbols and it is polluted with 10% noise (s20n10), whereas the fourth is similar to the third, but with 30% noise (s20n30). The realistic dataset , named VolosItineraries, includes the mobility of 210 vehicles travelling in the road network of the city of Volos (Greece) with different mobility patterns. The traffic simulations are conducted with SUMO and the trace files are injected into our custom simulator in order to perform prediction. Vehicles follow one of three different predefined routes, having a random velocity with a mean value of 11 m/s and a variation of 5 m/s. By using big variation in vehicle

velocities and by recording the position of each vehicle every T seconds (by default 5 seconds), we reassured that the final recorded trace for each vehicle is different from any other even if they follow the same path. Thus, each vehicle trace may contain repeating road segments representing along with the transition from one road segment to another, the staying on a road segment due to traffic congestion, road length and maximum velocity limit. The resulting alphabet created from the realistic dataset consists of several dozen different symbols. The total simulation time is one hour.

3) Performance measures: We use three measures to quantify performance. The first, is the prediction accuracy, which represents the percentage of correct predictions per 1,000 symbols of the input sequence. The second, is the processing time (milliseconds), in the form of the maximum time needed (worst case scenario) for a single prediction in a 1,000 symbol subsequence of the input sequence; it portrays the applicability of each competing algorithm in a real world ITS. The third is the number of trie nodes entered into the trie per 1000 symbols of the input sequence; it is an abstract measure of the memory footprint independent of any implementation.

4) Location update techniques: We update the movement history of a vehicle whenever it crosses the boundaries of a new road segment (with the use of the GPS technology) and every T seconds (by default 5 seconds). With our method:

- We ensure that all distinct road segmIDs will be recorded; missed road segmIDs effect on movement history is like noise in the symbol sequence, it disrupts the continuity of the symbols and affects negatively the repetition of the symbol strings.
- We differentiate each vehicle's movement patterns based on its habitual duration of stay in a road segmID.
- We control the amount of information entered in the system.

B. Evaluation of the results

The simulations were run on a PC with Intel core 2 duo 1.7 MHz CPU, 2GB main memory, 80GB hard disk 7200 rpm hard disk and MSWindows 7 64bit. The codes of the competing algorithms were compiled in Matlab R2015a. On the other hand, a typical communications box supporting Dedicated Short-Range Communications (DSRC), such as those commercialized by DELPHI runs on an x86 architecture Intel core 2 duo 2 GHz CPU, with 2GB onboard DDR2 RAM, and onboard 8 GB Solid State Disk. Therefore, our algorithms can run on an industrial onboard unit.

1) Tuning the RDM: We investigated the impact of blending the models of all orders and using different inter record time settings on RDM's performance.

a) The depth factor parameter: It enables RDM to exploit only a limited number of lower-order models during the blending strategy [2]. Practically, when excluding some of the lower-order models we bias the system to predict faster. However, we expect that happening at no cost to the RDM's performance, because the excluded models impact on the final probability assignment is suppressed due to the escape factor. In Fig. 4 we observe that RDM exhibits best performance in terms of prediction accuracy when we set the depthFactor equal to 1. A setting equal to 1 means that during the probability calculations we use only the order model in which the predictor currently lies and the immediate previous one (larger values for this parameter imply exploitation of smaller order models). In general, when depthFactor$>$ 1 the performance remains (almost) the same, as observed also in [18], however when a performance degradation exists it is due to past vehicle mobility patterns (not concerning the present vehicle movement) that are taken into account in the probability calculations.

	Depth Factor										
Dataset	0	1	2	3	4	5	6	7	8	9	10
s4n0	0.9834	0.9835	0.9832	0.9831	0.9831	0.9831	0.9831	0.9831	0.9831	0.9831	0.9831
s4n20	0.9433	0.9433	0.9431	0.9431	0.9431	0.9431	0.9431	0.9431	0.9431	0.9431	0.9431
s20n10	0.7181	0.7184	0.7184	0.7184	0.7184	0.7184	0.7184	0.7184	0.7184	0.7184	0.7184
s20n30	0.3650	0.3650	0.3645	0.3613	0.3613	0.3613	0.3613	0.3613	0.3613	0.3613	0.3613
Volositin	0.8466	0.8466	0.8466	0.8466	0.8466	0.8466	0.8466	0.8466	0.8466	0.8466	0.8466

Fig. 4. Impact of depthFactor parameter on RDM's prediction accuracy.

b) The inter record time - T: It influences decisively the frequency of the prediction process. We conducted several experiments w.r.t the VolosItineraries dataset to explore the impact of various inter record time settings on the performance of RDM and the results are presented in Fig. 5. In our work we set T = 5 sec as a tradeoff between RDM's prediction accuracy performance (Fig. 5 top plot) and the growth rate of its trie (Fig. 5 bottom plot). We avoid using settings of T $<$ 5 sec and T $>$ 10 sec because they miss to provide the predictions in a timely manner with each road segment change (which is a requirement for an online predictor). Note that the respective performance of RDM when T = 1 sec is deceptive as it is a consequence of the plethora of symbols that enter in the system in short time. A setting of T = 10 sec is also avoided as it leads to a larger trie (see Fig. 5 bottom plot). Event though that with a setting of T = 5 sec the RDM's trie is almost 1.7 times the respective trie of T = 30 sec, this is acceptable for the resource-rich VANET environment that RDM is planned to work.

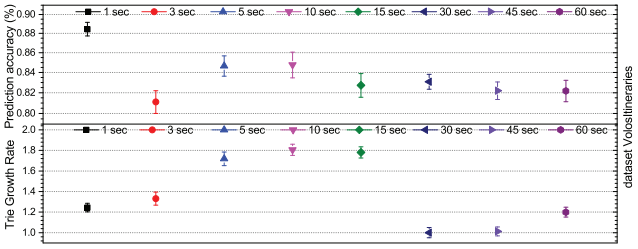

Fig. 5. Impact of Inter Record Time on RDM's performance

2) Comparison of the competing algorithms: The top plot in Fig. 6 depicts the performance of the competitors regarding their prediction accuracy for the first dataset (s4n0); RDM achieves almost 99% accuracy on the average whereas its competitors have inferior performance and they stagnate to a prediction accuracy of 70% at most. We expect that algorithms will achieve fast convergence and very good performance

because the alphabet is small and the trajectories are noiseless (existence of very few and strong patterns). The bottom plot in Fig. 6 shows the performance of the algorithms regarding their prediction accuracy for the second dataset (s4n20), which is noisier. RDM still exhibits very good performance (around 95% on the average and a maximum of 98.5%), however, now it is able to achieve more than 90% prediction accuracy only after consuming 4,000 symbols (contrast this to a 95% prediction accuracy after consuming only 2,000 symbols for the previous plot) because of the noise. The other algorithms performance is around 45% (on the average). Therefore, the rich dictionary of RDM and its new prediction mechanism make RDM to be always better than its competitors, and the performance gap widens (from 50% to 120% on the average) when noise is introduced.

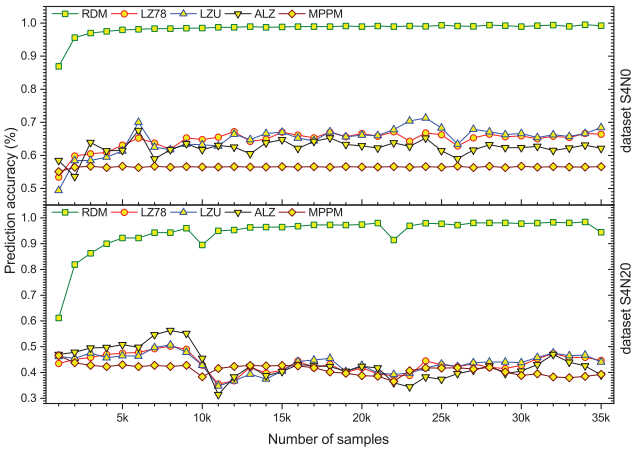

Fig. 6. Prediction accuracy (%) for small alphabets.

In the next experiment we increase the size of the alphabet, and evaluate the competitors under moderate (s20n10) and heavier (s20n30) noise. The results are illustrated in the two plots of Fig. 7. We expect that all algorithms performance will degrade significantly because now the alphabet is larger. Indeed, RDM achieves 72% and 37% performance for these datasets (contrast this to 99% and 95% accuracy in the previous two plots). Moreover, the distortion of the patterns due to the introduction of noise decisively affects RDM's performance, i.e., the prediction accuracy drops from 72% to 37% for s20n10 and s20n30, respectively. This drop in performance happens because RDM's deterministic prediction model cannot be widely exploited and thus it consults mainly the probabilistic model. However, RDM maintains the relative performance gap with its competitors (50% - 60% better). As expected, the performance gap between RDM and its competitors can not be as high as before, due a) to the larger alphabet, and b) to higher noise percentage (10% and 30% versus 0% and 20%), which collectively destroys the repetitive movement patterns.

Fig. 8 (top plot) depicts the performance of the competitors regarding their prediction accuracy for the VolosItineraries dataset. RDM if the best performing argorithm; it converges

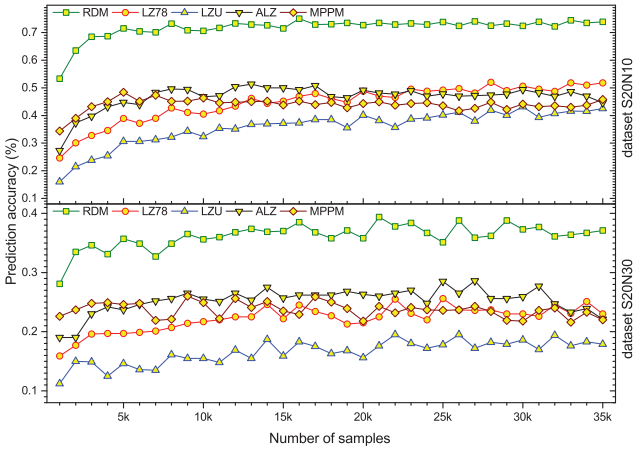

Fig. 7. Prediction accuracy (%) for larger and noisier alphabets.

to a prediction accuracy of 80% after having consumed only 2.000 symbols and achieves a mean prediction accuracy of 85% and a maximum of 90%. The performance of the other competitors is an increasing function with respect to the number of symbols that enter in the system. However they need to consume at least 5.000 symbols in order to converge to a prediction accuracy of 50%. MPPM is the second best performing algorithm with a mean prediction accuracy of 63% and a maximum of 76%. The rest of the competitors present a prediction accuracy of 57%, on the average. Therefore this realistic case reaffirms the earlier results, however now the performance gap with the second best performing algorithm (MPPM) narrows to 35%. In sum, we have shown in all the plots that RDM can achieve high prediction accuracy.

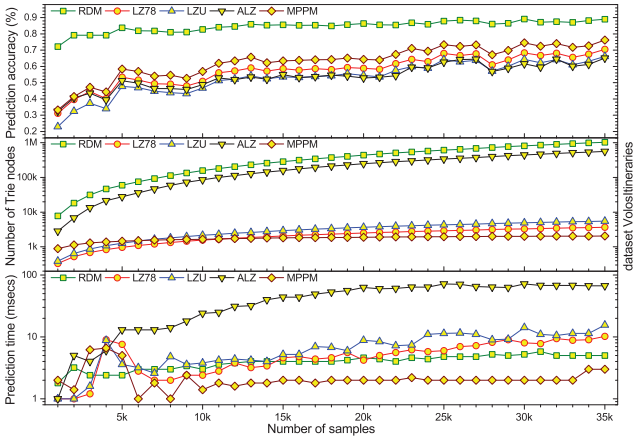

Fig. 8. Competing algorithms efficiency for the VolosItineraries dataset.

The middle plot of Fig. 8 depicts the memory footprint of the algorithms in the form of node count evolution in each respective trie as each symbol of the VolosItineraries dataset is processed. It can be seen that the pace of trie growth is logarithmic with respect to the sequence size for all the competitors except for MPPM where its trie stops growing after the parsing of 10,000 symbols as a consequence of

the fixed depth frequency tree of $k^{max} = 5$, it constructs. As expected, RDM constructs the largest trie which was our initial goal so as to achieve improved prediction accuracy. MPPM constructs thesmallest trie followed by LZ78 and LZU because of the small dictionary they have. It is noteworthy that the RDM's trie is the double size the respective ALZ's trie and two orders of magnitude larger than the LZUs and LZ78s respective tries. Moreover, RDM and ALZ trie grows much faster, whereas LZU trie size also starts growing quickly but stop increasing so fast at a lower level. It is noteworthy that the large trie that RDM produces does not lead to significant communication overhead. In an operational VANET environment a single packet can accommodate a large number of trie nodes, and in any case, if two vehicles need to exchange their mobility profile, they will not exchange the full trie, but rather only the part of it (a few branches) that correspond to the particular area where they are both moving at that particular time.

The bottom plot of Fig. 8 depicts the per-symbol maximum prediction time needed (worst case in a 1,000 symbol sequence) for the VolosItineraries dataset. Intuitively someone would expect the largest dictionary be the most time-consuming to process, however RDM succeeds in being faster than the majority of its competitors and competitive to MPPM, because of the hybrid mechanism it employs for prediction. It takes 6 milliseconds (worst case) for the RDM to deliver predictions, which is fast enough to support real-time applications, such as vehicular safety applications. On the other hand, MPPM requires the least processing effort which is due to the fixed depth ($k^{max} = 5$) frequency tree that it constructs. Overall, RDM is the best performing predictor; it constructs the largest trie, and it is not the slowest algorithm. MPPM in general exhibits similar performance with ALZ except for the realistic dataset (where it is better) thereby confirming the results in [13] and for the s4N0 dataset (where it is worse). ALZ is better than LZ78 in terms of accuracy for large alphabets and input sequences with a small length, thus confirming the results in [6]. It also emerges that ALZ is not significantly better than LZ78 in the other cases, thereby confirming the findings in [14], but contradicting the findings in [6], because we used the exclusion strategy. ALZ converges faster than LZ78, which supports the results in [6] and all the predictors tries are growing at a logarithmic pace (except MPPM), which mirrors the behavior reported in [14]. Finally, the creation of RDM addresses a comment that appeared in [18] stating that there is a significant gap that needs be filled by the improvement of online Markov predictors.

VII. Conclusions

Accurate prediction of vehicular trajectories in a VANET environment is an essential mechanism for ITS. The prediction methods based on Markov predictors are particularly appealing, because of their generality and prediction accuracy. Tailored for the resource-rich VANET environments, the proposed RDM Markov predictor remains highly efficient in terms of prediction accuracy and per-symbol maximum prediction time. RDM is likely to be of immediate interest to ITS providers. By maintaining global dictionaries along with individual vehicle profile decoded from updates, it will be possible to predict group behavior. In VANETs, this can lead to better traffic management, and more efficient bandwidth management and quality of service (QoS) in p2p applications. As future work we plan to bound the growth of the frequency tree that RDM produces and integrate RDM with novel VANET routing protocols that can be used in safety or ecorouting applications.

References

[1] S. A. Bagloee, M. Tavana, M. Asadi, and T. Oliver. Autonomous vehicles: challenges, opportunities, and future implications for transportation policies. *Journal of Modern Transportation*, 24(4):284–303, 2016.

[2] A. Bhattacharya and S. K. Das. LeZi-Update: An information-theoretic framework for personal mobility tracking in PCS networks. *Wireless Networks*, 8(2):121–135, 2002.

[3] E. Cayirci and I. F. Akyildiz. Optimal location area design to minimize registration signaling traffic in wireless systems. *IEEE Transactions on Mobile Computing*, 2(1):76–85, 2003.

[4] M. Chaqfeh, A. Lakas, and I. Jawhar. A survey on data dissemination in vehicular ad hoc networks. *Vehicular Communications*, 1:214–225, 2014.

[5] M. Feder, N. Merhav, and M. Gutman. Universal prediction of individual sequences. *IEEE Transactions on Information Theory*, 38:1258–1270, 1992.

[6] K. Gopalratnam and D. J. Cook. Online sequential prediction via incremental parsing: The Active LeZi algorithm. *IEEE Intelligent Systems*, 22(1):52–58, 2007.

[7] A. Houenou, P. Bonnifait, V. Cherfaoui, and W. Yao. Vehicle trajectory prediction based on motion model and maneuver recognition. In *IEEE RSJ*, pages 4363–4369, 2013.

[8] J. Huang and H.-S. Tan. Vehicle future trajectory prediction with a DGPS/INS-based positioning system. In *IEEE ACC*, 2006.

[9] O. Kaiwartya and S. Kumar. Geocast routing: Recent advances and future challenges in vehicular adhoc networks. In *IEEE SPIN*, pages 291–296, 2014.

[10] D. Katsaros and Y. Manolopoulos. Prediction in wireless networks by Markov chains. *IEEE Wireless Communications*, 64:56–64, 2009.

[11] T. Kimura, T. Matsuda, and T. Takine. Probabilistic store-carry-forward message delivery based on node density estimation. In *IEEE CCNC*, pages 432–437, 2014.

[12] P. Kolios, V. Friderikos, and K. Papadaki. Future wireless mobile networks. *Vehicular Technology*, 6(1):24–30, 2011.

[13] S. K. Pulliyakode and S. Kalyani. A modified ppm algorithm for online sequence prediction using short data records. *IEEE Communications Letters*, 19(3):423–426, 2015.

[14] A. Rodriguez-Carrion, C. Garcia-Rubio, and C. Campo. Performance evaluation of lz-based location prediction algorithms in cellular networks. *IEEE Communications Letters*, 14(8):707–709, 2010.

[15] C. U. Saraydar, O. E. Kelly, and C. Rose. One-dimensional location area design. *IEEE Transactions on Vehicular Technology*, 49:1626–1632, 2000.

[16] M. Schreier, V. Willert, and J. Adamy. Bayesian, maneuver-based, long-term trajectory prediction and criticality assessment for driver assistance systems. In *IEEE ITSC*, pages 334–341, 2014.

[17] R. Simmons, B. Browning, Z. Yilu, and V. Sadekar. Learning to predict driver route and destination intent. In *IEEE ITSC*, pages 127–132, 2006.

[18] L. Song, D. Kotz, R. Jain, and X. He. Evaluating next-cell predictors with extensive wi-fi mobility data. *IEEE Transactions on Mobile Computing*, 5(12):1633–1649, 2006.

[19] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24:530–536, 1978.

[20] A. Y. Xue, R. Zhang, Y. Zheng, X. Xie, J. Huang and Z. Xu. Destination Prediction by Sub-Trajectory Synthesis and Privacy Protection Against Such Prediction. In *IEEE ICDE*, 2013.

[21] A. Vahedian, X. Zhou, L. Tong, Y. Li, and J. Luo. Forecasting Gathering Events Through Continuous Destination Prediction on Big Trajectory Data. In *ACM SIGSPATIAL ICAGIS*, pages 1–10, 2017.