HarmonyMoves: A Unified Prediction Approach for Moving Object Future Path

Mohammed Abdalla[1], Hoda M. O. Mokhtar[2], Neveen ElGamal[3]
Faculty of Computers and Artificial Intelligence
Beni-Souef University[1]
Cairo University[2,3]

***Abstract*—Trajectory prediction plays a critical role on many location-based services such as proximity-based marketing, routing services, and traffic management. The vast majority of existing trajectory prediction techniques utilize the object's motion history to predict the future path(s). In addition to, their assumptions that the objects' moving with recognized patterns or know their routes. However, these techniques fail when the history is unavailable. Also, these techniques fail to predict the path when the query moving objects lost their ways or moving with abnormal patterns. This paper introduces a system named *HarmonyMoves* to predict the future paths of moving objects on road networks without relying on their past trajectories. The system checks the harmony between the query object and other moving objects, after that if the harmony exists, this means that there are other objects in space moving like the query object. Then, a Markov Model is adopted to analyze this set of similar motion patterns and generate the next potential road segments of the object with their probabilities. If the harmony does not exist, *HarmonyMoves* considers this query object as abnormal object (*object lost the way and needs support to return back known routes*), for this purpose *HarmonyMoves* employed a new module to handle this case. A fundamental aspect of *HarmonyMoves* lies in achieving a high accurate prediction while performing efficiently to return query answers.**

Keywords*—Trajectory prediction; machine learning; moving objects

I. Introduction

Location-based services that consider the future paths and locations of moving objects proved to be essential in several daily life activities. Uses of location-based services includes proximity-based marketing; in which companies push ads notifications only to customers within the same geographic region, travel information; in which the user can be provided by real-time information, such as traffic updates or weather reports and plan trips accordingly, traffic management; in which drivers can predict the congested traffic regions.

Several techniques have been proposed to predict the possible future paths and destinations of moving objects and explores their results in better understanding of the human mobility into location-based services [3], [9], [10], [24], [26]. Overall, most of the existing prediction techniques depend basically on the trajectories of the moving objects' saved on the system beforehand to be able to predict the future path and destinations. Nevertheless, these techniques suffer from one or more weaknesses. These techniques (1) fail to predict future paths and destinations when the query object's history does not exist during the model training, (2) some of these techniques consider assumptions like moving objects following the shortest path or following preferred routes to reach the needed destination or moving in linear movements, which usually fails and turns to be non logical in many scenarios, (3) some of these techniques don't consider the objects that lost their ways or moving with unrecognized motion patterns, (4) suffer from efficiency and technical limitations.

This paper introduces a novel system named *HarmonyMoves* proposed to provide an efficient future path prediction in case that the query object's historical motions are absent, in addition to guiding the query objects that lost their ways to follow the correct paths and reach to their destinations in a safely manner.

The idea of *HarmonyMoves* is to identify if there are other moving objects currently in space moving like the query object or not. If the system finds similar objects moving like the query object, the system considers the query object as normal object, and trained for similar objects trajectories to anticipate the possible future path of the query object. In case that the system identifies that there are no other objects moving like the query object, the system considers the query object as anomalous , and starts to guide the query object in terms of the nearest moving object in vicinity.

Contributions. The main contributions of this research are the following:

1) This work addresses the prediction of moving objects' future path in case that their past trajectories are absent.
2) This work introduces a novel similarity function to help all types of moving objects to predict their routes and find their final destinations, in addition to, in case of normal objects this function leads to a significant increase in prediction efficiency by removing irregularities in the input data and make the prediction model trains on data similar to each other.
3) We devise a novel algorithm to handle moving objects that lost their paths and need to predict future path or reach final destinations.

The rest of this paper is organized as follows. Section 2 studies related work and investigates different directions in the area of trajectory prediction. Section 3 formally defines the problem. The architecture of the *HarmonyMoves* system along with its basic components is illustrated in Section 4. Section 5 experimentally evaluates *HarmonyMoves*. Finally, Section 6 concludes the paper.

II. Related Work

This section surveys the previous studies which predict the future paths of the moving objects. Overall, this section covers two major related directions; namely path prediction and predictive search queries.

A. Path Prediction

Different studies have been introduced to predict future paths of moving objects [2], [5], [6], [9], [17], [21], [27]–[30]. However, some of these studies predict the complete paths [5], [6], [27], others predict partial paths [2], [17], [28], [29], whereas some predict the final destination [9], [21], [30].

For predicting the complete path of the end-users, in [6], a similarity trip algorithm is developed to predict the end-to-end route of a vehicle based on vehicle's past trips observations. The proposed algorithm matches the first part of a driver's current trip with one of the set of previously observed trips. In [27], authors develop a driving path prediction technique that employed Hidden Markov Model (HMM). This technique predicts accurately a vehicle's entire path as early in a trip's lifetime as possible without knowing origins and destinations in advance. Authors in [5] propose a novel algorithm for predicting a driver's route based on a probabilistic prediction of the driver's destination. The algorithm is based on only one parameter that shows how efficiently a driver drives. When this parameter is computed, the algorithm does not need to store a history of drivers' trips, and it works in places a driver has never visited.

For predicting the partial path of the end-users, in [29], authors propose a technique which adopted Hidden Markov Model to predict future road segments from the complete driving path. The authors in their solution neglects the traffic conditions. In contrast, authors in [17], [28] take into their account traffic conditions that change dynamically and prove that the work introduced in [29] is not adequate to capture variable order Markov dependencies. Some studies integrate semantic information of routes to improve prediction. Authors in [2] propose an approach to improve partial route prediction by considering semantic information associated with routes such as day and time of departure.

For predicting the finial destinations of the end-users, in [9], [30], prediction models are proposed to detect the trajectory patterns that occurred frequently and utilize these patterns to predict the most potential destination of the moving objects. Authors in [21] introduce a technique predict the final destination of vehicle trips based on their initial partial trajectories. The technique starts by clustering of trajectories that describes user behaviour. Then, it models main traffic flow patterns by a mixture of 2d Gaussian distributions.

B. Predictive Queries

According to the study in [19], a predictive search query is conducted as follows: "*a query that retrieves the set of moving objects' which will intersect a query window during a future time interval*". Several studies have been in this direction [4], [7], [16], [19], [20], [25].

In [7], *Panda* system is presented, *Panda* aims to support long-term query prediction. In [16], a novel index structure, named Predictive tree (P-tree) is proposed for processing predictive queries against moving objects on road networks. Efficient techniques are introduced in [4], [20], [25] that try to optimize the performance of predictive window queries. In [1], the authors introduce the iRoad framework for evaluating predictive queries on moving objects for road networks. In [8], authors review the current research trends and present their related applications in the field of predictive spatiotemporal queries processing.

TABLE I. Moving Objects Trajectories

Trajectory	Road Segments
τ_1	e_6, e_7, e_{16}, e_{14}
τ_2	e_8, e_{16}, e_{14}
τ_3	e_7, e_{16}, e_{14}
τ_4	e_7, e_{16}, e_{19}
τ_5	e_7, e_{16}, e_{17}
τ_6	e_8, e_{16}, e_{17}
τ_7	e_{11}, e_9, e_7, e_{16}
τ_8	e_{18}, e_{13}, e_{12}

HarmonyMoves distinguishes itself from the above researches in the following: (1) *HarmonyMoves* is considered as a novel learning model which performs the prediction without any prior knowledge of the query object's motions history, (2) *HarmonyMoves* proves efficient and accurate results by providing novel similarity approach which neglects irrelevant input features from prediction process, (3) *HarmonyMoves* is considered as first approach that handles moving objects which loses their ways, and (4) *HarmonyMoves* produces accurate results compared with other state-of-art methods.

III. Preliminaries

This section explores the main terminologies that will be used through the rest of the paper and then highlights the problem definition.

A. Definitions

Definition 1: Road network graph G(N, E) consists of a set of nodes N that represent road intersections, and a set of edges E that represent road segments.

Fig. 1 presents an illustrative example of a road network graph. Table I contains a sample of objects' trajectories τ_1 to τ_{14} moving over the given road network graph(G).

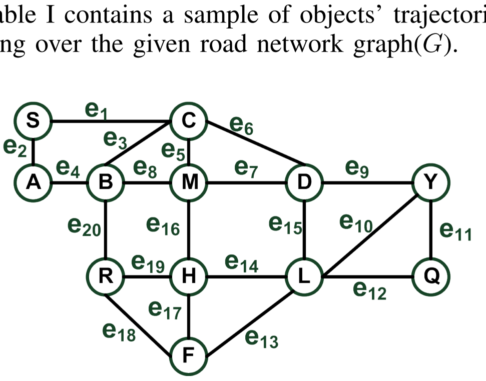

Fig. 1. Road Network Graph

Definition 2: Trajectory τ is an ordered sequence of edges travelled by a moving object, i.e., τ={e_1, e_2, e_3,....e_n}

Definition 3: Future road segments $\mathcal{F}$ is defined as how many future road segments query object aims to predict.

Definition 4: Harmony it represents the total *Similarity* value between the query object's trajectory Q_τ and another moving object's trajectory τ_i. The equation 1 presents the harmony computing. The Harmony value is computed by getting the maximum length between the query object's trajectory Q_τ and another moving object's trajectory τ_i, then computes the edit distance between (Q_τ,τ_i). After that the result of edit distance is subtracted from maximum length between the query object's trajectory Q_τ and another moving object's trajectory τ_i. Finally, the value subtracted is divided by maximum length between the query object's trajectory Q_τ and another moving object's trajectory τ_i. The edit distance is computed based on Levenshtein distance metric [15], this metric measures the difference between two sequences. It computes the total number of edits (insertions, deletions or substitutions) needed to change one sequence into the other.

$$Harmony = \frac{MaxLength(Q_\tau, \tau_i) - E_{Distance}(Q_\tau, \tau_i)}{MaxLength(Q_\tau, \tau_i)} \tag{1}$$

Definition 5: Prediction confidence $\mathcal{P}$ is a probability value that reveals a certainty degree which the query object will travel through the predicted path. The higher the value, the more accurate the prediction is.

Definition 6. Number of Previously Travelled Segments (Prediction Order) λ reveals to the last number of segments moved by query object's trajectory over the road network. For example: assume $\tau_i = \{e_1, e_2, e_3, e_4\}$ and λ =2, this means that the extracted segments will be (e_3, e_4).

B. Problem Statement

Assume a query object's trajectory Q_τ moving in its current trip and a set of another objects $Trajectories_{Current}$ moving in their current trips. Let Q_τ needs to predict the future path, in case that its own historical movements data is unavailable or inaccessible. In addition to, Q_τ needs to predict the future path when it moves with unknown motion patterns or lost their ways.

The aim of this research is to accurately predict the future paths of the query object and perform efficiently by returning the query responses in a reasonable time.

The significant of this research lies in solving critical cases such as; (1) when a query object does not own previous trips to be used for prediction purposes or the query object is not allowed to use the his past trips for security concerns, and (2) when a query object lost his way or moving with unknown motion patterns and needs to predict the future path while there is no other objects moving like him.

IV. System Architecture

This section presents the proposed approach (*HarmonyMoves*) for predicting the future path of the query object.

Main Idea. The main idea of *HarmonyMoves* is to anticipate the possible future paths or destinations and returns them to the query object. The proposed solution starts by measuring the similarity, which in this work named as Harmony, between the query object and other objects moving currently in the same road network.Next, the system checks if there are other objects moving like the query object (*Harmony¿0*), the system contains a module named *"normal objects handler module"* to deal with this case, this module starts by getting the similar trajectories, after that utilizes these trajectories to generate predictions by using a learning model. It is important to note that predictions generated is a map (key/value) pairs, which composed of travelled road segments as a key and possible future segments as a value. Finally, after the predictions generated, the system query the created map by the last segments moved by the query object and returns the possible future segments. Next, if the system checks the harmony and discovers that there are no other objects moving like the query object (*Harmony=0*), the system considers the query object as a vagrant moving object. Vagrant objects represent the objects that lost their ways or objects moving with unknown roads needs to reach their destinations or predict next segments to reach to the known routes. More specifically, the system contains a module named *"vagrant objects handler module"* to deal with this type of objects, first the system the system tracks the nearest moving objects and based on his trajectory, the system suggests a possible route to reach the destination or suggests multi next road segments.

Fig. 3 describes the architecture of the proposed solution *HarmonyMoves* which composed of three main components namely, the Harmony Checker Module, Normal Objects Handler Module, and Vagrant Objects Handler Module. *HarmonyMoves* receives as input the moving objects' trajectories and the query object's trajectory, and returns the query object's future path or destination.

Algorithm. Algorithm **??** illustrates the pseudo code of *HarmonyMoves*. The algorithm receives, (a) the current query object's trajectory, (b) other moving objects' trajectories currently in the road network, (c) prediction order, and (d) a number of future road segments required to be predicted. As output, the algorithm returns the query object's predicted path or possible destination. The algorithm has three main steps that are briefly described as follows:

Step1: Harmony checker. This step checks the similarity between the query object and other moving objects, in line 4 the algorithm checks the harmony between the query object and other objects currently moving with the query object in the road network. The proposed module that accomplishes this job is explained later in Section IV-A.

Step2: Handling Normal Query Objects. The objective of this step is to handle the prediction process in case that the query object has similar objects currently moving like it currently in the system. The proposed module that accomplishes this job is explained later in Section IV-B.

Step3: Handling Vagrant Query Objects. The objective of this step is to handle the prediction process in case that the query object has no similar objects currently moving like it currently in the system. The proposed module that accomplishes this job is explained later in section IV-C.

A. Harmony Checker Module

Main Idea. The main idea of the Harmony Checker Module is to extract trajectories set of objects that currently move same as the query object. Then, the module checks if this set of trajectories is empty, the *HarmonyMoves* considers the query object as vagrant object, otherwise it considers it as normal one. For this purpose, the harmony checker module calculates the harmony score between the query object's trajectory and all other trajectories of objects that are currently moving in the space by utilizing proposed equation 1. If the Harmony value is equal to 0, the Harmony Checker Module considers the query object is vagrant, otherwise it considers it as normal one.

Example. Assume that the query object's trajectory is τ_7, and assume that the other objects currently moving in road network are τ_2 e_8, e_{16}, e_{14}, and τ_3 e_7, e_{16}, e_{14}. The module starts by calculating the harmony between the query object τ_7 and τ_2,τ_3. For τ_2, the module first gets the max length between τ_7 and τ_2 which is 3, then computes the edit distance between τ_7 and τ_2 which is 1. As a result, the module subtracted the edit distance score from max length between τ_7 and τ_2 divided by the same length, so the final result is Harmony $= \frac{1}{3}$. Similarly, the system computes the harmony between τ_7 and τ_3 which is Harmony $= \frac{2}{3}$. Finally, the system considers τ_7 as normal query object. For second case, assume that the query object τ_8 and other moving objects are τ_7 and τ_2, the harmony in all case will be equal to 0 and the system considers τ_7 as vagrant object.

B. Normal Objects Handler Module

This section presents a new module that handles the cases when the query object have other objects moving similar like it in the system. This module builds a predictions model from the list of similar trajectories and, then returns the query object's predicted path. The module composed of two main sub modules namely, Predictions Builder Module, and Path Prediction Module.

1) Predictions Builder Module: Main Idea. The input to this module is the set of similar trajectories that are moving same as the query object. The output of this module is a hash-map. This map composed of Key and value pairs, where the key is an ordered set of previous segments (already-visited) in a trajectory and the value is a sequence of future (want to visit) segments and its probability $\mathcal{P}$. The prediction function employed in this module is developed based on notions of Hidden Markov Model(HMM) [14]

Hidden Markov Model (HMM). The Markov Model represents the sequence of moved segments as a sequence of *X(i)*, where i is the offset of the segemnt in the order they are encountered inside the trajectory. As stated in [11], HMM refers to the trajectory edges as ..., X(-2), X(-1), X(0), X(1), X(2),..., where X(0) is the object's current road edge. X(-1) and X(-2) refer to one and two previous steps, respectively. X(1) and X(2) refer to the unknown future road segments to be predicted. HMM produces a probabilistic prediction over future road segments based on the past moved segments. For example, $\mathcal{P}$[X(1)] represents the probability of a one segment ahead of the object's current location. $\mathcal{P}$[X(2)] is the probability of two segments ahead, and so on. The first order Markov model says that the probability $\mathcal{P}$[X(1)] for the next road segment is independent of all the object's previous history except for X(0), the current segment:

$$\mathcal{P}[X(1)|X(0), X(-1), X(-2), ...] = \mathcal{P}[X(1)|X(0)] \qquad (2)$$

Similarly, a second order Markov model will depend on the two most recent edges,$\mathcal{P}$[X(1)|X(-1),X(0)]. Overall, a n^{th} order Markov model, ($n \geq 1$), can be created to anticipate the m^{th} future edge ($m \geq 1$). The general n^{th} order model can be expressed as following:

$$\mathcal{P}_n[X(m)] = \mathcal{P}[X(m)|X(-n+1), X(-n+2), ..., X(0)] \qquad (3)$$

Fig. 2 descries the results produced by this module.

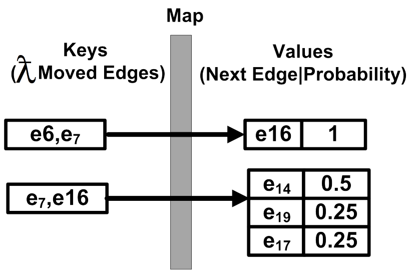

Fig. 2. Predictions Builder Module Results

2) Path Prediction Module: Main Idea. The main idea of this module is to return the possible future segments to the query object. First, the module detects the prediction order configured, then based on the value of prediction order, the module gets the last segments moved by the query object and considers it as key. After that, the module query the created map by the key and return the future path.

Example. Let's the query object is τ_7, assume the value configured for prediction order λ=2, and the number of future steps needs to be predicted $\mathcal{F}$ is 1. According to this setup, the module constructed the key as (e7,e16) and then query the map created as per Figure 2. The path prediction Module returns next segments to the query object, first, $\{e_{14}$,$\mathcal{P} = 0.5\}$,$\{e_{19}$,$\mathcal{P} = 0.25\}$, and $\{e_{17}$,$\mathcal{P} = 0.5\}$

C. Vagrant Objects Handler Module

This section presents a new module that handles the cases when the query object has no similar objects moving like it in the system.

Main Idea. The main idea behind this module is to handle the prediction process in case that there is no similar trajectories currently moved in the system anymore and the query object issue a query asking for the future path. The paper named these objects as Vagrant Moving objects. Vagrant objects represent the objects that lost their ways or objects moving with unknown roads need to reach their destinations. Handling these objects is significant as most of road accidents

Algorithm 1 *HarmonyMoves:* Algorithm

```
1: INPUT: Query object's trajectory $Q_\tau$, Moving objects' current trajectories $Trajectories_{Current}$, Future steps $\mathcal{F}$
2: SET $SimilarTrajectoriesList\ S \leftarrow \phi$
3: /* Step 1: compute harmony */
4: $S$ = Compute_Harmony($Q_\tau$,$Trajectories_{Current}$)
5: /* Step 2: checks $Q_\tau$ is vagrant object or normal one */
6: if $S \neq \phi$ then
7:     Call Normal Objects Handler Module
8: else
9:     Call Vagrant Objects Handler Module
10: end if
11: OUTPUT: Return $QueryResult$
```

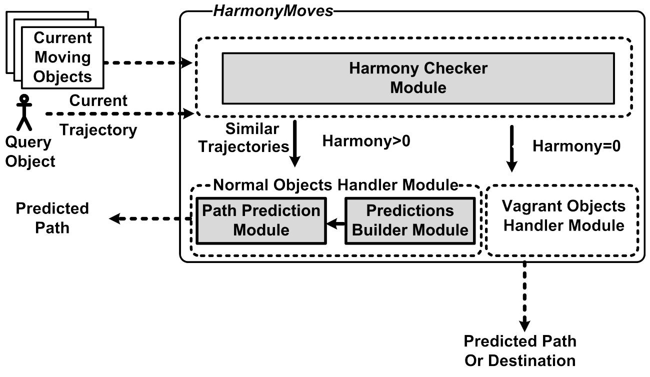

Fig. 3. *HarmonyMoves* Architecture

Algorithm 2 Unknown Destinations Handler

```
1: procedure GET_FUTURE_PATH
2: INPUT: Query object's trajectory $Q_\tau$, $Future\ Steps\ \mathcal{F}$, Moving object's trajectories $Trajectories_{Current}$, Distance $d$
3: OUTPUT: Return $Predicted\_Path$
4:     /* Set compared Trajectory */
5:     SET $Compared_\tau \leftarrow \phi$
6:     /* Last edge in query object trajectory */
7:     SET $Q(CurrentEdge)_\tau \leftarrow \phi$
8:     /* edges connected direct to query object current node */
9:     SET $edgesList \leftarrow \phi$
10:    /* All expected paths */
11:    SET $PathList \leftarrow \phi$
12:    if $\mathcal{F} = 0$ then
13:        Exit and return $Predicted\_Path$
14:    end if
15:    for i=0 to $\mathcal{F}$ do
16:        $Compared_\tau$ = call R-Tree to get nearest Trajectory to $Q_\tau$ from $Trajectories_{Current}$ within $d$
17:        $Q(CurrentEdge)_\tau$ = Current edge of $Q_\tau$
18:        Put in edgesList edges connected by $Q(CurrentEdge)_\tau$
19:        for each edge $e_i$ in $edgesList$ do
20:            FetchedPath= $Q_\tau$ + $e_i$
21:            Add $FetchedPath$ to $PathList$
22:        end for
23:        for each $FetchedPath$ in $PathList$ do
24:            ComputeSimilarity($FetchedPath$,$Compared_\tau$)
25:            Set Most Similar FetchedPath as $Q_\tau$
26:            Set Most Similar FetchedPath as $Predicted\_Path$
27:        end for
28:    end for
29:    Get_Future_Path($Q_\tau, \mathcal{F} - 1, Trajectories_{Current}$)
30: end procedure
```

and people deaths coming from similar cases. This module handles 2 possible scenarios; (1) vagrant objects don't know their destinations and need to update their paths per time, and (2) vagrant objects know their destinations and don't know how to reach. For unknown destinations, the module starts by using R-Tree [22] to get the nearest object's trajectory within a specific distance. After that, the module considers this trajectory as a compared trajectory. Next, the module identifies the current edge of the query object, fetches the next connected edges to this current edge and creates paths. The module iterates over these created paths and computes the similarity between each iterated path and the compared trajectory by using the Hausdorff [23] algorithm. As a result, the module considers the most similar path to the compared trajectory as new query object trajectory and make recursive calls until predicting all the required future steps. For known destinations, the module starts by using R-Tree to get the nearest object's trajectory within a specific distance. After that, the module considers this trajectory as a compared trajectory. Next, the module identifies the current edge of the query object, fetches the next connected edges to this current edge and create paths. The module iterates over these created paths and computes the similarity between each iterated path and the compared trajectory by using the Hausdorff algorithm. As a result, the module considers the most similar path to the compared trajectory as new query object trajectory and make recursive calls until the query object reach to its destination.

1) Unknown Destinations: This section presents the algorithm and illustrative example for case 1; vagrant objects don't know their destinations and need to update their paths per time.

Algorithm. Algorithm 2 illustrates the pseudo code of the proposed Handler Module that handle case 1 defined before for vagrant objects. The algorithm takes four input parameters, (a) Query object's trajectory Q_τ, (b) $Future\ Steps\ \mathcal{F}$, (c) Moving object's trajectories $Trajectories_{Current}$, and (d)Distance d. In line 15, the algorithm starts by iterating over the number of future steps $\mathcal{F}$. In each iteration, the algorithm gets the compared trajectory from the objects currently moved with the query object by using the R-tree within specific distance, line 16. In line 17, the algorithm identifies the current edge of the query object, obtains the edges directly connected to the current node and saved them in $edgesList$. Then, the algorithm iterates over the $edgesList$, and for each iterated edge, the algorithm append this edge to the query object trajectory, then consider this as new path. Each path is saved to $PathList$ and algorithm considers it as fetched path. After that, in line 23, the algorithm iterates over the $PathList$, and computes the similarity between each path and the compared trajectory. Hence, the algorithm makes the most similar path as the predicted path. Besides this, the algorithm considers it as a new query object's trajectory, (lines 23-27). Then, algorithm decrease the number of future steps by 1 and make a recursive calls each time by the new calculated future steps, new created query object's trajectory. Finally, The algorithm check if the $\mathcal{F} = 0$, it skips and returns back the predicted path (Lines 12-14).

Example. Assume that the query object's trajectory is $\tau_7(e_{11}, e_9, e_7, e_{16})$, assume the number of future steps to be predicted ($\mathcal{F}$) is 2, assume that $SimilarTrajectoriesList$ S is ϕ, and assume d is 1 Kilo Meter. First, the algorithm gets the compared trajectory $Compared1_\tau$, say for example (e_3, e_1, e_2). Second, gets the connected edges to the current node of the query object which are e_{14}, e_{19}, e_{17}. Accordingly, the algorithm creates 3 paths: $path_1\{e_{11}, e_9, e_7, e_{16}, e_{14}\}$,$path_2\{e_{11}, e_9, e_7, e_{16}, e_{19}\}$, and $path_3\{e_{11}, e_9, e_7, e_{16}, e_{17}\}$. After that, the algorithm computes the similarity between each path and $Compared1_\tau$, as result algorithm returns $path_2$ as predicted path. Algorithm checks if $\mathcal{F}$=0, it returns no, so it makes a recursive call by the new created query object which equal to $path_2\{e_{11}, e_9, e_7, e_{16}, e_{19}\}$, and the $\mathcal{F}$=1. Then, algorithm identifies the new compared trajectory, assume is $Compared2_\tau(e_3, e_1, e_2)$. Next, the algorithm gets the connected edges to the current node of the query object which are $\{e_{20}, e_{18}\}$. Accordingly, the algorithm creates 2 paths: $path_1\{e_{11}, e_9, e_7, e_{16}, e_{19}, e_{20}\}$,$path_2\{e_{11}, e_9, e_7, e_{16}, e_{19}, e_{18}\}$. Then, the algorithm computes the similarity between each path and $Compared2_\tau$, as a result, the algorithm returns $path_1\{e_{11}, e_9, e_7, e_{16}, e_{19}, e_{20}\}$ as predicted path. The algorithm checks if $\mathcal{F}$=0, it returns yes, so it exit and returns $path_1\{e_{11}, e_9, e_7, e_{16}, e_{19}, e_{20}\}$ as expected path

2) Known Destinations: This section presents the algorithm and illustrative example for case 2; vagrant objects know their destinations and need to reach them.

Algorithm. Algorithm 3 illustrates the pseudo code of the proposed Handler Module that handle case 2 defined before for vagrant objects. The algorithm takes four input parameters, (a) Query object's trajectory Q_τ, (b) Query object's destination $Dest$, (c) Moving object's trajectories $Trajectories_{Current}$, and (d)Distance d. In line 12, in each iteration, the algorithm gets the compared trajectory from the objects currently moved with the query object by using the R-tree within specific distance. In line 13, the algorithm identifies the current edge of the query object, obtains the edges directly connected to the current node and saved them in $edgesList$. Then, the algorithm iterates over the $edgesList$, and for each iterated edge, the algorithm append this edge to the query object trajectory, then consider this as new path. Each path is saved to $PathList$ and algorithm considers it as fetched path. After that, in line 19, the algorithm iterates over the $PathList$, and computes the similarity between each path and the compared trajectory. Hence, the algorithm makes the most similar path as the predicted path. Besides this, the algorithm considers it as a new query object's trajectory, (lines 19-23). Finally, the algorithm checks if the current edge of the query object is same as the destination required, if yes the algorithm returns the predicted path; otherwise, the algorithm makes a recursive calls until reach to its destination (Lines 24-28).

Example. Assume that the query object's trajectory is $\tau_7(e_{11}, e_9, e_7, e_{16})$, assume the destination needs to be predicted e_{20}, assume that $SimilarTrajectoriesList$ S is ϕ, and assume d is 1 Kilo Meter. First, the algorithm gets the compared trajectory $Compared1_\tau$, say for example (e_3, e_1, e_2). Second, gets the connected edges to the current node of the query object which are e_{14}, e_{19}, e_{17}. Accordingly, the algorithm creates 3 paths: $path_1\{e_{11}, e_9, e_7, e_{16}, e_{14}\}$,$path_2\{e_{11}, e_9, e_7, e_{16}, e_{19}\}$, and $path_3\{e_{11}, e_9, e_7, e_{16}, e_{17}\}$. After that, the algorithm computes the similarity between each path and $Compared1_\tau$, as result algorithm returns $path_2$ as predicted path. Algorithm checks if the query object's current edge equal to e_{20}, it returns no, so it make a recursive call by the new created query object which equal to $path_2\{e_{11}, e_9, e_7, e_{16}, e_{19}\}$. Then, algorithm identifies the new compared trajectory, assume is $Compared2_\tau(e_3, e_1, e_2)$. Next, the algorithm gets the connected edges to the current node of the query object which are $\{e_{20}, e_{18}\}$. Accordingly, the algorithm creates 2 paths: $path_1\{e_{11}, e_9, e_7, e_{16}, e_{19}, e_{20}\}$,$path_2\{e_{11}, e_9, e_7, e_{16}, e_{19}, e_{18}\}$. Then, the algorithm computes the similarity between each path and $Compared2_\tau$, as a result, the algorithm returns $path_1\{e_{11}, e_9, e_7, e_{16}, e_{19}, e_{20}\}$ as predicted path. The algorithm checks if if the query object's current edge equal to e_{20}, it returns yes, so it exit and returns $path_1\{e_{11}, e_9, e_7, e_{16}, e_{19}, e_{20}\}$ as expected path.

Algorithm 3 known Destinations Handler

```
1: procedure GET_FUTURE_PATH
2: INPUT: Query object's trajectory $Q_\tau$, Query object's destination $Dest$, Moving object's trajectories $Trajectories_{Current}$, Distance $d$
3: OUTPUT: Return $Predicted\_Path$
4:     /* Set compared Trajectory */
5:     SET $Compared_\tau \leftarrow \phi$
6:     /* Last edge in query object trajectory */
7:     SET $Q(CurrentEdge)_\tau \leftarrow \phi$
8:     /* edges connected direct to query object current node */
9:     SET $edgesList \leftarrow \phi$
10:    /* All expected paths */
11:    SET $PathList \leftarrow \phi$
12:    $Compared_\tau$ = call R-Tree to get nearest Trajectory to $Q_\tau$ from $Trajectories_{Current}$ within $d$
13:    $Q(CurrentEdge)_\tau$ = Current edge of $Q_\tau$
14:    Put in edgesList edges connected by $Q(CurrentEdge)_\tau$
15:    for each edge $e_i$ in edgesList do
16:        FetchedPath= $Q_\tau$ + $e_i$
17:        Add $FetchedPath$ to $PathList$
18:    end for
19:    for each $FetchedPath$ in $PathList$ do
20:        ComputeSimilarity($FetchedPath$,$Compared_\tau$)
21:        Set Most Similar FetchedPath as $Q_\tau$
22:        Set Most Similar FetchedPath as $Predicted\_Path$
23:    end for
24:    if $Q(CurrentEdge)_\tau = Dest$ then
25:        Exit and return $Predicted\_Path$
26:    else
27:        Get_Future_Path($Q_\tau, Dest, Trajectories_{Current}$)
28:    end if
29: end procedure
```

V. Experiments

This section evaluates experimentally the proposed $HarmonyMoves$ system.

A. Experimental Setup

Data-sets. All the experiments introduced in this work use real trajectories data-set collected by Microsoft Research team in the Geo-life project [13] covering the period between

April 2007 to August 2012. Furthermore, these trajectories are splitted into small trajectories with an average length of each one 10 road segments. Thus, the total number of trajectories is 5000. All trajectories GPS data points (longitude, latitude) are map-matched to road segments along the road network. The road network data is captured from OpenStreetMap [18]. Road network data in this work represents Hamilton city in the USA. The map-matching algorithm is out the scope of this paper [12].
Experimental Settings. The prediction function employed inside $HarmonyMoves$,R-Tree module, and hausdorff distance functions are implemented using Java with JDK 1.9 inside eclipse PHOTONID IDE. All experiments are accomplished on a PC with Intel(R) core(tm) i7-4770 cpu @3.40ghzM, and running on the linux ubuntu operating system.

B. HarmonyMoves Evaluation and Results

We next report our findings

Exp 1:- Comparison between HarmonyMoves and other baseline models. In this set of experiments, Fig. 4 compares 2 baseline prediction techniques of by $HarmonyMoves$. This is set of experiments choose RandomGuess and RMF for comparison. RMF is an HMM model-based path prediction method , which computes motion function to capture movements. This experiment proves that the proposed system $HarmonyMoves$ outperform other methods in prediction accuracy. The justification behind this is the $HarmonyMoves$ depends on its work on novel similarity function the prune irregularities in input data before the training process.

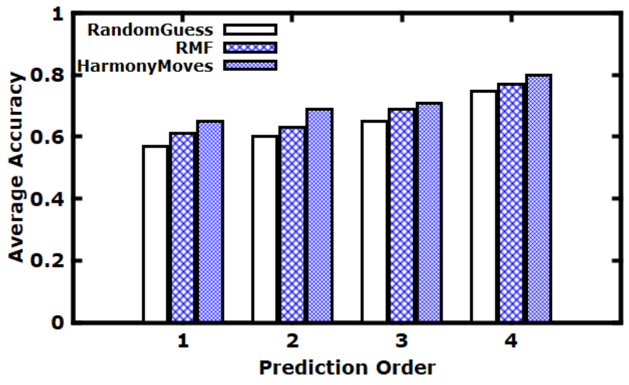

Fig. 4. Comparison between baseline methods

Exp 2:- Accuracy evaluation of vagrant objects VS. normal objects.This set of experiments investigates the impact of increasing number of moving objects on both vagrant objects and normal objects. Fig. 5 shows that the normal objects always achieve higher prediction accuracy more than vagrant objects. The justification behind this is that the normal objects have more objects moving similar like them, and the prediction done through high similarity value, as result prediction in efficient and easily manner.

Exp 3:- Impact of increasing similar objects on normal query objects. This set of experiments studies the impact of increasing number of similar moving objects on the normal

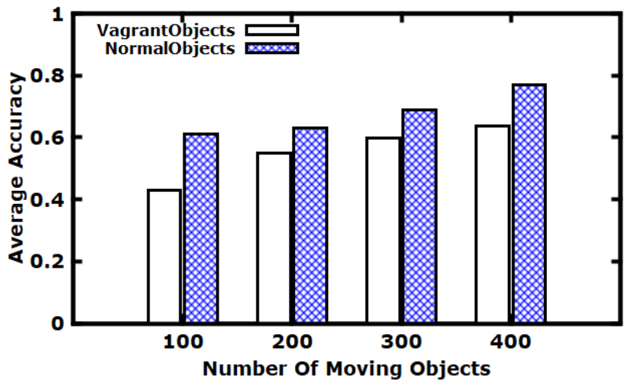

Fig. 5. Vagrant Objects VS. Normal Objects

query objects. It is observed from Fig. 6 that the increase in similar moving objects number, the total increase on the average prediction accuracy. The justification behind this is that the increased number of similar objects will make the prediction HMM model trains on more input data similar to each other and make the model perform efficiently.

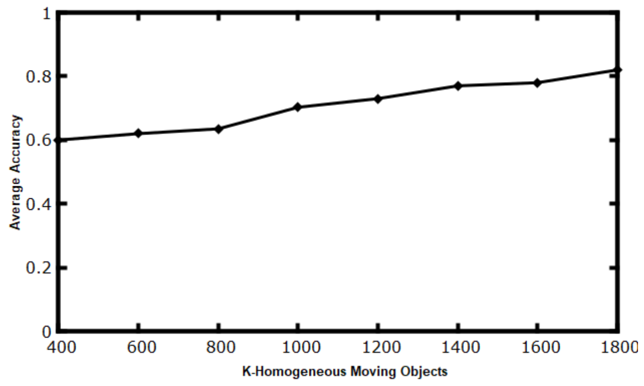

Fig. 6. Impact of increasing similar objects on Normal query objects

Exp 4:- Impact of increasing number of future steps on normal and vagrant objects. In this set of experiments, Fig. 5 compares the normal objects with the vagrant objects according to the CPU processing time when they are need to predict the future steps. It is observed that normal objects consumes less CPU time than vagrant objects during the prediction process. The justification behind this is that the normal objects move similar like other objects and the hash-map created during the prediction can contain cashed values before which can be accessed by O(1). Additionally, vagrant objects make many recursive calls and need each time to identify compared trip to predict next road or final destination.

VI. Conclusion

In this paper, we presented the *HarmonyMoves* system that predicts the future trajectory of a moving query object when the object's movement history or past trajectories is absent. In addition, *HarmonyMoves* helps moving objects that lost their ways to go their destinations or follow recognized routes. The

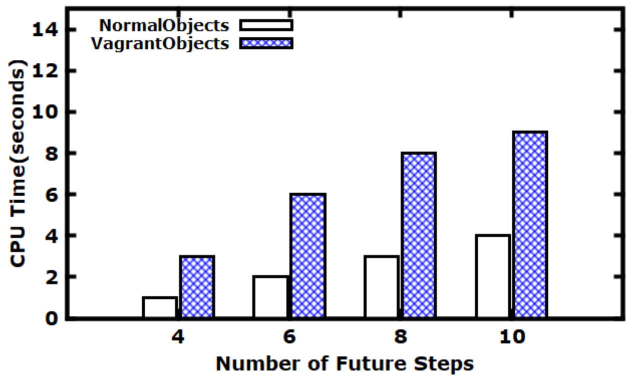

Fig. 7. Impact of increasing number of future steps

system leverages the harmony between the query object and other moving objects that are existing in the space at the same time, explores similarities in their trajectories. If the harmony exist, *HarmonyMoves* utilizes these similarities to predict the query object's future paths. Otherwise, *HarmonyMoves* try to get the nearest object within specific distance to the query object and the then utilize the nearest object's trajectory to guide object that lost their ways. The system composed of three main components; Harmony Checker Module, Normal Objects Handler Module, and Vagrant Objects Handler Module. Experiments showed that *HarmonyMoves* provided accurate prediction results and achieved high performance.

In the future work, we plan to make *HarmonyMoves* able to handle huge number of end-users who submit several predictive queries on the cloud-based through big data frameworks, like Apache-Spark. This makes *HarmonyMoves* more generic and scalable framework.

REFERENCES

[1] Abdeltawab M. Hendawi, Jie Bao, and Mohamed F. Mokbel. iRoad: A Framework For Scalable Predictive Query Processing On Road Networks. *In Proceedings of the International Conference on Very Large Databases, VLDB 2013, Riva Del Garda, Italy*, August, 2013.

[2] Francisco Neto, Cláudio Baptista, and Claudio Campelo. Prediction of Destinations and Routes in Urban Trips with Automated Identification of Place Types and Stay Points. *Brazilian Journal of Cartography*, 2016.

[3] Chaoming Song, Zehui Qu, Nicholas Blumm, and Albert-László Barabási. Limits of predictability in human mobility. *Science*, 327(5968), 2010.

[4] Rui Zhang, H.Jagadish, Bing Dai, and Kotagiri Ramamohanarao. Optimized algorithms for predictive range and KNN queries on moving objects. *Information Systems*, 35(8), 2010.

[5] John Krumm,Robert Gruen,and Daniel Delling. From destination prediction to route prediction. *Journal of Location Based Services*, 2013.

[6] Jon Froehlich and John Krumm. Route Prediction from Trip Observations. *SAE Technical Paper Series*, 2008.

[7] Abdeltawab Hendawi and Mohamed Mokbel. Panda. *Proceedings of the 20th International Conference on Advances in Geographic Information Systems-SIGSPATIAL 12*, 2012.

[8] Abdeltawab Hendawi and Mohamed Mokbel. Predictive Spatio-Temporal Queries: A Comprehensive Survey and Future Directions. *In Proceeding of the first ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems, MobiGIS 2012, Redondo Beach, CA,*, November 2012.

[9] Heng Shen, Hoyoung Jeung, Qing Liu and Xiaofang Zhou. A Hybrid Prediction Model for Moving Objects. *2008 IEEE 24th International Conference on Data Engineering*, 2008.

[10] Hassan Karimi and Xiong Liu. A predictive location model for location-based services. *In Proceedings of the ACM-GIS'03*, 2003.

[11] John Krumm. A Markov Model for Driver Turn Prediction. *SAE Technical Paper Series*, 2008.

[12] John Krumm, Julie Letchner, and Eric Horvitz. Map Matching with Travel Time Constraints. *SAE Technical Paper Series*, 2007.

[13] Yu Zheng, Xing Xie, Wei-Ying Ma. Geolife: A collaborative social networking service among user, location and trajectory. *IEEE Data Eng. Bull.*, 33(2):32–39, 2010.

[14] Irmtraud Meyer. Hidden Markov Model (HMM, Hidden Semi-Markov Models, Profile Hidden Markov Models, Training of Hidden Markov Models, Dynamic Programming, Pair Hidden Markov Models). *Dictionary of Bioinformatics and Computational Biology*, 2004.

[15] Klaus U.Schulz, Stoyan Mihov. Fast string correction with Levenshtein automata. *International Journal on Document Analysis and Recognition*, 2002.

[16] Abdeltawab Hendawi, Jie Bao, Mohamed Mokbel, and Mohamed Ali. Predictive Tree: An Efficient Index for Predictive Queries on Road Networks. *2015 IEEE 31st International Conference on Data Engineering*, 2015.

[17] Emilian Necula. Dynamic traffic flow prediction based on GPS Data. *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*, 2014.

[18] OpenStreetMap. https://www.openstreetmap.org/, accessed on april 2019.

[19] Yufei Tao, Dimitris Papadias and Jimeng Sun. The TPR*-tree: an optimized spatio-temporal access method for predictive queries. *Proceeding VLDB '03 Proceedings of the 29th international conference on Very large data bases*, 2003.

[20] Katerina Raptopoulou, Apostolos Papadopoulos, and Yannis Manolopoulos. Fast Nearest-Neighbor Query Processing in Moving-Object Databases. *GeoInformatica*, 7(2):113–137, 2003.

[21] Philippe C.Besse,Brendan Guillouet,Jean-Michel Loubes,and Francois Royer. Destination Prediction by Trajectory Distribution Based Model. *IEEE Transactions on Intelligent Transportation Systems*, pages 2470–2481, 2018.

[22] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider and Bernhar Seeger. The R*-tree: an efficient and robust access method for points and rectangles. *ACM SIGMOD Record*, 19(2):220–231, 1990.

[23] B. Sendov. Hausdorff Distance. *Hausdorff Approximations*, pages 23–48, 1990.

[24] Hoda Mokhtar, Jianwen Su, and Oscar Ibarra. Map Matching with Travel Time Constraints. *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems - PODS 02*, 2002.

[25] Xiaofeng Xu, Li Xiong, Vaidy Sunderam, and Yonghui Xiao. A Markov Chain Based Pruning Method for Predictive Range Queries. *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems-GIS 16*, 2016.

[26] J. Sun, Dimitris Papadias, Yufei Tao, and Bin Liu. Querying about the past, the present, and the future in spatio-temporal databases. *Proceedings. 20th International Conference on Data Engineering*, 2004.

[27] Ning Ye,Zhong-Qin Wang,Reza Malekian,Qiaomin Lin,and Ru-Chuan Wang. A Method for Driving Route Predictions Based on Hidden Markov Model. *Mathematical Problems in Engineering*, 2015.

[28] Yisong Zheng, Qian Wang, Wangling, and Mo Kuang. Research into the driver's route choice under existing real-time traffic information. *2008 IEEE International Conference on Industrial Engineering and Engineering Management*, 2008.

[29] R. Simmons, B. Browning, Yilu Zhang, and V. Sadekar. Learning to Predict Driver Route and Destination Intent. *2006 IEEE Intelligent Transportation Systems Conference*, 2006.

[30] Shaojie Qiao, Nan Han, William Zhu, and Louis Gutierrez. TraPlan: an effective three-in-one trajectory-prediction model in transportation networks. *IEEE Transactions on Intelligent Transportation Systems*, 16(3), 2015.