Querying Spatio-Temporal Patterns in Mobile Phone-Call Databases

Marcos R. Vieira [1#], Enrique Frías-Martínez [⋆], Petko Bakalov [‡]
Vanessa Frías-Martínez [⋆], Vassilis J. Tsotras [#]

[#] *University of California, Riverside, CA – USA*
{mvieira,tsotras}@cs.ucr.edu

[‡] *ESRI, Redlands, CA – USA*
pbakalov@esri.com

[⋆] *Telefónica Research, Madrid – Spain*
{efm,vanessa}@tid.es

***Abstract*— Call Detail Record (CDR) databases contain many millions of records with information about mobile phone calls, including the users' location when the call was made/received. This huge amount of spatio-temporal data opens the door for the study of human trajectories on a large scale without the bias that other sources, like GPS or WLAN networks, introduce in the population studied. Furthermore, it provides a platform for the development of a wide variety of studies ranging from the spread of diseases to planning of public transportation. Nevertheless, previous work on spatio-temporal queries does not provide a framework "flexible" enough for expressing the complexity of human trajectories. In this paper we present Spatio-Temporal Pattern System (STPS) to query spatio-temporal patterns in very large CDR databases. STPS uses a regular-expression query language that is intuitive and that allows for any combination of spatial and temporal predicates with constraints, including the use of variables. The design of the language takes into consideration the layout of the areas being covered by the cellular towers, as well as "areas" that label places of interested (e.g. neighborhoods, parks, etc). A full implementation of the STPS is currently running with real, very large CDR databases at Telefónica Research Labs. An extensive performance evaluation of the STPS shows that it can efficiently find very complex mobility patterns in large CDR databases.**

I. Introduction

The recent adoption of ubiquitous computing technologies by very large portions of the world population has enabled – for the first time in human history – to capture large scale spatio-temporal data about human motion. In this context, mobile phones play a key role as sensors of human behavior since they are typically owned by one individual that carries it at (almost) all times and are nearly ubiquitously used. Hence, it is no surprise that most of the quantitative data about human motion has been gathered via Call Detail Records (CDRs) of cell phone networks.

When a cell phone makes/receives a phone call the information regarding the call is logged in the form of a CDR. This information includes, among others, originating and destination phone numbers, the time and date when the call started, and the towers used, which gives an approximation of the caller's/callee's geographical location. Such data is very rich and has been used recently for several applications, such as to study user's social networks [1], [2], [3], human mobility behaviors [4], [5], and cellular network improvement [6].

The volume of data generated by a given operator in the form of CDRs is huge, and it contains valuable spatio-temporal information at different levels of granularity (e.g. citywide, statewide, nationwide, etc). This information is relevant not only for telecommunication operators but also as a base for a broader set of applications with social connotations like commuting patterns, transportation routes, concentrations of people, modeling of virus spreading, etc. The ability to efficiently query CDR databases to search for spatio-temporal patterns is key to the development of such applications. Nevertheless, the commercial systems available cannot efficiently handle this kind of spatio-temporal processing. One possible solution to search for such patterns is to perform a sequential scanning of the *entire* CDR database and, for each user, check whether it qualifies using a subsequence matching-like algorithm (e.g. KMP (Knuth-Morris-Pratt) [7]). Such naive approach however is computationally extremely expensive due to the amount of users/CDRs to be processed. Furthermore, there is the fact that no information about the temporal dimension of the pattern (e.g. within given time frame) or spatial properties (e.g. in a given neighborhood) can be specified.

Taking into consideration the large volume of data and current implementation of the CDR storage systems for telecommunication providers, one effective way to support such spatio-temporal pattern queries is to extend the current systems with some indexes and algorithms to efficiently process such queries. One aspect that has to be considered is that commercial storage systems are in their majority implemented on top of Relational Database Management System (RDBMS). Therefore the provided solution should use the available RDBMS infrastructure such as tables, indexes (e.g. inverted indexes and B-trees), merge-join algorithms, and so on.

In this paper we present the Spatio-Temporal Pattern System (STPS) to query spatio-temporal patterns in CDR databases.

[1]Work done while author was an intern at Telefónica Research–Spain. M. Vieira's Ph.D. work has been funded by a CAPES/Fulbright fellowship.

The STPS allows users to express mobility pattern queries with a regular expression-like language that can include "variables" in the pattern specification. Variables serve as "placeholders" in the pattern for *explicit* spatial regions and their value is determined during the pattern query evaluation. An example for a query with variables is the pattern "find users who visited the same mall twice in the last 24 hours". In this scenario we do not know in advance which one is the mall visited by the user. So we use variables which can take values from the set of malls to specify the user behavior in a pattern query. We have to pay attention that in the above example the variable should appear *twice* in the pattern.

STPS also includes lightweight index structures that can be easily implemented in most commercially RDBMS. We present an extensive experimental evaluation of the proposed techniques using two large, real-world CDR databases. The experimental results reveal that the proposed STPS framework is scalable and efficient under several scenarios tested. Our proposed system is up to 1,000 times faster than a base line implementation, making the STPS a very robust approach for querying and analyzing very large phone-call databases. A fully operational prototype of the system is implemented and running at Telefónica Research Labs.

This paper presents a continuation of our previous work in pattern query evaluation in trajectorial archives [8]. In [8] we proposed a regular-expression based language and evaluation algorithms to query patterns in trajectorial archives. In this paper we adopt that approach and study its application in the domain of CDR databases. In particular, we modified the join-based evaluation algorithm to handle trajectories specified in CDR format rather than the traditional form, defined as sequence of object locations with their *longitude* and *latitude* coordinates. This change in the data format poses changes in the query languages as well. In [8] the query language includes several query predicates that are well suited when the exact location of the object is known for a continuous period of time. An example of such a predicate is the *distance-based predicate* used to find trajectories that passed *as close as possible* to some area of interest. In a CDR database however, the exact location of the mobile user is unknown and users are not continuously monitored. Thus, the pattern language proposed here is more suitable for CDR databases (e.g. cells, user defined areas, temporal predicates to track *hopping* during a call or for different calls, etc). Our language proposed in this paper also supports user defined constraints (e.g. conditions, inequalities, time constraints, etc). Furthermore, the query evaluation system is redesigned to work with the features (e.g. tables, B^+-trees and so on) of a commercially available RDBMS, since CDR databases are typically implemented in such systems.

The remainder of this paper is organized as follows: Section II discusses the related work; Section III provides some basic descriptions on the infrastructure; Section IV provides the formal description of the STPS language; the proposed system is described in Section V and its experimental evaluation appears in Section VI; Section VII concludes the paper.

II. Related Work

Infrastructures for querying spatio-temporal patterns have already been studied in the literature in different contexts, mainly for: (1) time-series databases; (2) similarity between trajectories; and (3) single predicate for trajectory data (GPS).

Pattern queries have been used in the past for querying time series using SQL-like query language [9], [10], or event streams using a NFA-based method [11]. Our work differs from those solutions mainly because it provides a richer language to specify spatio-temporal patterns and an efficient way to evaluate them. For moving object data, patterns have been examined in the context of query language and modeling issues [12] as well as query evaluation algorithms [13].

Similarity search among trajectories has been also well studied. Work in this area focuses on the use of different distance metrics to measure the similarity between trajectories (e.g. [14], [15], [16]).

Single predicate queries for trajectory data, like Range and *NN* queries, have been well studied in the past (e.g. [17]). In these contexts, a query is expressed by a single range or NN predicate. To make the evaluation process more efficient, the query predicates are typically evaluated utilizing hierarchical spatio-temporal indexing structures [18]. Most structures use the concept of Minimum Bounding Regions (MBR) to approximate the trajectories, which are then indexed using traditional spatial access methods, like the MVR-tree [19]. These solutions, however, are focused only on single predicate queries and further constructions to build a more complex query, e.g. a sequence of combination of both predicates, are not supported. In [13] an incremental ranking algorithm for simple spatio-temporal pattern queries is presented. These queries consist of range and *NN* predicates specified using only *fixed* regions. Our work differs in that we provide a more general and powerful query framework where queries can involve both fixed and *variable* regions as well as topological operators, temporal predicates, constraints, etc., and an explicit ordering of the predicates along the temporal axis.

In [20] a *KMP*-based algorithm [7] is used to process patterns in trajectorial achieves. This work, however, focuses only on the *contain* topological predicate and cannot handle *explicit* or *implicit* temporal ordering of predicates. Furthermore, this approach on evaluating patterns is effectively reduced to a sequential scanning over the list of trajectories stored in the repository: each trajectory is checked individually, which becomes prohibitive for large trajectory archives. We show in Section VI that this approach is very inefficient.

III. Infrastructure for Data Acquisition

Cell phone networks are built using a set of Base Transceiver Stations (BTS) that are in charge of communicating mobile phone devices with the cell network. The area covered by a BTS is called a cell. A BTS has one or more directional antennas (typically two or three, covering 180 or 120 degrees, respectively) that define a sector and all the sectors of the same BTS define the cell. At any given moment in time, a cell phone is covered by one or more antennas.

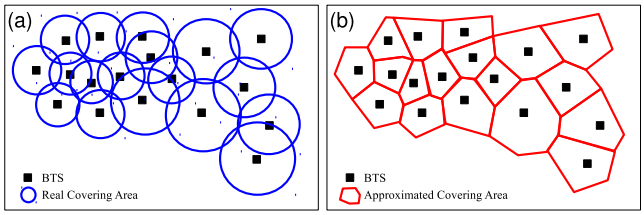

Fig. 1. (a) Original coverage areas of BTSs and (b) approximation of coverage areas by Voronoi diagram.

TABLE I

A SET OF CDRS REPRESENTING 4 DIFFERENT CALLS.

timestamp	*dur*	$phone^o_{id}$	$phone^d_{id}$	BTS^o_{id}	BTS^d_{id}	type
1123001	3	4324542	4333434	231	121	V
1123004	2	4324542	4333434	232	435	V
1123006	5	4324542	4333434	234	121	V
1123235	2	4324542	5334212	235	231	V
1123237	4	4324542	5334212	231	233	V
1124113	3	4333434	4324541	238	231	V
1124116	4	4333434	4324541	239	231	V
1124116	1	5334212	4333434	451	239	S

Depending on the network traffic, the phone selects the BTS to connect to. The geographical area covered by a cell depends mainly on the power of individual antennas. Depending on the population density, the area covered by a cell ranges from less than 1 Km^2, in dense urban areas, to more than 5 Km^2, in rural areas. Each BTS has latitude/longitude attributes that indicate its location, a unique identifier BTS_{id}, and the polygon representing its cell. For simplicity, we assume that the cell of each BTS is a 2-dimensional non-overlapping region, and we use Voronoi diagrams to define the covering areas of the set of BTSs considered. Figure 1(a) presents a set of BTSs with the original coverage for each cell, and (b) the simulated coverage obtained using Voronoi diagrams. While simple, this approach gives us a good approximation of the coverage area of each BTS. In practice, to build the "real" diagram of coverage, one has to consider several factors in the mobile network, e.g. power and orientation of each antenna.

CDR databases are populated when a mobile phone, connected to the network, makes/receives a phone call or uses a service in the network (e.g., SMS, MMS, etc.). In the process, the information regarding the time and the BTS where the user was located when the call was initiated is logged, which gives an *indication* of the user's geographical location at a given period in time. Note that no information about the *exact* user's location inside a cell is known. Furthermore, for a given call it is possible to store not only the initial BTS during the period of a call, but also all BTSs used during it in case caller/callee move to other cells in the network (*hopping*). The STPS supports this richer representation of the users' mobility.

The following attributes from CDR databases are used in the STPS system: **(1)** the originating phone number $phone^o_{id}$; **(2)** the destination phone number $phone^d_{id}$; **(3)** the type of service (voice: V, SMS: S, MMS: M, etc.); **(4)** the BTS identifier used by the originating number (BTS^o_{id}); **(5)** the BTS identifier used by the destination number (BTS^d_{id}); **(6)** *timestamp* (date/time) of the connection between $phone^o_{id}$ and $phone^d_{id}$ in BTS^o_{id} and BTS^d_{id}, respectively; and **(7)** the duration *dur* while $phone^o_{id}$ and $phone^d_{id}$ connected to BTS^o_{id} and BTS^d_{id} (*hopping* enabled), respectively. Since in the STPS we are only interested in users' mobility, we do not make any distinctions between caller and callee. Therefore, the superscript symbols (^{o} and ^{d}) in $phone_{id}$ and BTS_{id} are omitted in the STPS language and framework. The BTS identifier is only known for $phone_{id}$ that are clients of the telecommunication operator keeping the CDR database. When the *hopping* is enabled, a new CDR row is created every time either users connects to different BTS_{id} during the same phone call, otherwise, a single CDR is stored to represent the initial position of $phone^o_{id}$ and $phone^d_{id}$ for the total duration of the call *dur*.

Table I shows a set of CDRs for 4 distinct calls. In this example the BTS *hopping* option is enabled. Phone number 4324542 makes a phone call to 4333434 starting in BTS^o_{id}=231 at timestamp 1123212. Then the user 4324542 moves from BTS^o_{id}=231 to BTS^o_{id}=232 3 minutes after starting the call, generating another record in the database. After 2 minutes, user 4324542 moves to BTS^o_{id}=234 staying there for 5 minutes. The user 4333434 is connected to BTS^d_{id}=121, then to 435, and then back to 121 during the call. When a user is connected to a particular BTS_{id}, it does not necessary mean that the user is on the same place for the whole period of connection. The second call represents the call made from 4324542 to 5334212, and the third one from 4333434 to 4324541. The eight entry of the table details an SMS sent from 5334212 to 4333434 when they were connected to BTS^o_{id}=451 and BTS^d_{id}=239, respectively. If the BTS *hopping* was not enabled, the first three entries would have been presented as a single one, with just the initial BTS^o_{id}=231 and a total duration of 10 minutes.

IV. THE STPS PATTERN QUERY LANGUAGE

We define a trajectory $T(phone_{id})$ of a mobile user with identifier $phone_{id}$ in CDR databases as a sequence of records $\{\langle phone_{id}, BTS_{id}, t_1, dur_1\rangle, \ldots, \langle phone_{id}, BTS_{id}, t_m, dur_m\rangle\}$, where BTS_{id} is the BTS identifier which serviced the mobile user $phone_{id}$ at timestamp t_i for the dur_i of time ($t_i, t_m \in \mathbb{N}$, $t_i < t_m$ and $dur_i \in \mathbb{N}$). This trajectory definition covers both formats described in the previous section: (i) as a sequence of BTSs where the user was connected to the mobile network; or (ii) as a sequence of a trajectory *segments* (at a BTS level) where each *segment* represents the movement of the user between two BTS during a phone call. We assume that CDRs using this representation are stored in an archive as shown in Figure 3(d).

The STPS language uses the above definition of a trajectory to covers both data formats; i.e. we can query for patterns using records for the same phone call or different calls. This is achieved by associating temporal predicates for each spatial predicate which can be used to restrict the user "movements" into a time frame of a single phone call. In the next subsections we describe in details the syntax of the STPS pattern

$$\mathcal{Q} := (\mathcal{S}\ [\bigcup \mathcal{C}])$$
$$\mathcal{S} := \{\mathcal{P}_1.\mathcal{P}_2.,...,.\mathcal{P}_n\}, |\mathcal{S}| = n$$
$$\mathcal{P}_i := \langle op_i, \mathcal{R}_i[, t_i]\rangle$$
$$op_i := \textit{disjoint}|\textit{meet}|\textit{overlap}|\textit{equal}|$$
$$\textit{inside}|\textit{contains}|\textit{covers}|\textit{coveredBy}$$
$$\mathcal{R}_i \in \{\Sigma \cup \Gamma\}$$
$$t_i := (t_{from} : t_{to}) \mid t_s \mid t_r$$

Fig. 2. The STPS Pattern Query Language.

query language and its components: the spatial predicates, the temporal predicates, and the set of spatio-temporal constraints.

A. STPS Language Syntax

A pattern query $\mathcal{Q}$ is defined as $\mathcal{Q} = (\mathcal{S}\ [\bigcup \mathcal{C}])$, where $\mathcal{S}$ is a sequential pattern and $\mathcal{C}$ is an optional set of spatio-temporal constraints. The set of constraints $\mathcal{C}$ is used to specify certain spatio and/or temporal constraints that an answer has to satisfy in order to be considered as part of the result. A trajectory with identifier $phone_{id}$ matches the pattern query $\mathcal{Q}$ if it satisfies both the sequential pattern $\mathcal{S}$ and the set of spatio-temporal constraints $\mathcal{C}$. A sequential pattern $\mathcal{S}$ is defined as a sequence of an arbitrary number n of spatio-temporal predicates $\mathcal{S} = \{\mathcal{P}_1.\mathcal{P}_2.,...,.\mathcal{P}_n\}$.

Each spatio-temporal predicate $\mathcal{P}_i \in \mathcal{S}$ is defined by a triplet $\mathcal{P}_i = \langle op_i, \mathcal{R}_i[, t_i]\rangle$, where op_i represents a topological relationship operator, $\mathcal{R}_i$ a spatial region, and t_i the optional temporal predicate. The operator op_i describes the topological relationship that the spatial region $\mathcal{R}_i$ and the coverage area of the BTS defining a trajectory with identifier $phone_{id}$ must satisfy over the (optional) temporal predicate t_i. Figure 4 details formally the syntax of the STPS language.

B. Spatial Predicates

A key part of our STPS language syntax is the definition of the spatial alphabet Σ, used in the spatio-temporal predicates $\mathcal{P}_i$. We choose the Voronoi diagram cells, that represent the covering areas of each BTS, to serve as "letters" in our alphabet Σ. This is because the BTS coverage areas represent the finest level of granularity in which the data is stored in CDR databases. In the rest of the paper we use capital letters to represent the set of BTS coverage areas in the system, e.g. $\Sigma = \{A, B, C, ...\}$. Such coverage areas can participate as spatial regions $\mathcal{R}_i$ in the definition of the spatio-temporal predicates $\mathcal{P}_i$.

The users however are not restricted to use only BTS coverage areas in their queries. On top of this BTS coverage partitioning the user can define its own geographical maps with different resolution and different types of regions (school districts, airports, shopping etc.). Also, users can define polygons defined by a set of latitude/longitude pairs to define a set of areas. All other regions, defined by the user, have to be approximated by set of coverage areas in the alphabet Σ. For instance, one can define the downtown area of a city by creating regions *DOWNTOWN* = $\{D, E, H\}$ and *STADIUM-1* = $\{S1\}$, where the downtown area is approximated by the union of the coverage areas of BTS D, E and H and the *Stadium-1* is approximated by the coverage area of BTS $S1$. The same BTS_{id} can be used in the definition of multiple regions and not all BTS have to be included in each geographical map.

Inside the spatial predicates $\mathcal{P}_i$ we use finite set of spatial regions $\mathcal{R}_i$. Those regions can be one of the following: (i) a particular $\text{BTS}_{id} \in \Sigma$; (ii) an alias $\mathcal{A}$ defined by a set of one or more $\text{BTS}_{id} \in \Sigma$; or (iii) a variable in Γ. We refer to the first two groups of spatial regions $\mathcal{R}_i$ as *predefined* spatial regions. A predefined region (i.e., $S1 \in \Sigma$) is explicitly specified by the user in the query predicate (e.g. "Stadium-1" *STADIUM-1* = $\{S1\}$ in our example). In contrary, the third group of spatial regions, termed *variable* spatial regions, references an arbitrary region in the map and it is denoted by a lowercase letter preceded by the "@" symbol (e.g. "@x"). A variable region is defined using symbols from the set $\Gamma = \{@a, @b, @c, ...\}$. Unless otherwise specified, a *variable* takes a single value (instance) from Σ (e.g. @a=C); however, in general, one can also specify in $\mathcal{C}$ the possible values of a specific *variable* as a subset of Σ (e.g., "any city district with museums"). Conceptually, *variables* work as placeholders for explicit spatial regions and can become instantiated (bound to a specific region) during the query evaluation in a process similar to unification in logical programming.

Moreover, the same *variable* "@x" can appear in several different predicates of pattern $\mathcal{S}$, referencing to the same region everywhere it occurs. This is useful for specifying complex queries that involve revisiting the same region many times. For example, a query like "@x.$S1$.@x" finds mobile users that started from some region (denoted by variable "@x"), then at some point passed by region $S1$ and then they visited the same region they started from.

We finish with the description of the last component of the spatial predicate: the topological relationship operator op_i. In this paper we use the eight topological relationships: *disjoint*, *meet*, *overlap*, *equal*, *inside*, *contains*, *covers* and *coveredBy* defined by [12]. Given a phone user record $\langle phone_{id}, \text{BTS}_j, t_i\rangle$ and a region $\mathcal{R}_i$, the operator op_i returns a boolean value whether the coverage area in the phone user record BTS_j and the region $\mathcal{R}_i$ satisfy the topological relationship op_i (e.g., an *Inside* operator will return value *true* if the user associated with $phone_{id}$ was serviced by BTS which has coverage area inside the spatial region $\mathcal{R}_i$. For simplicity in the rest of the paper we assume that the spatial operator is *Inside* and it is thus omitted from the query examples.

C. Temporal Predicates

As it was mentioned in the definition of the STPS language a spatio-temporal predicate $\mathcal{P}_i$ may include an explicit temporal predicates t_i. Those predicates can be in the form of: (a) time *interval* $(t_{from} : t_{to})$ where $t_{from} \leq t_{to}$ (for example "between 4pm and 5pm"); (b) time *snapshot* t_s (for example "at 3:35pm"); or (c) time *relative* $t_r = t_i - t_{i-1}$ from the time instance t_{i-1} when the previous spatio-temporal predicate $\mathcal{P}_{i-1}$ satisfied (for example "1 hour after the user left his home"). Those temporal predicates imply that the

spatial relationship op_i between BTS$_j$ and region $\mathcal{R}_i$ should be satisfied in the specified time frame t_i (e.g. "passed by area $S1$ between 4pm and 5pm"). If the temporal predicates is not specified, we assume that the spatial relationship can be satisfied *any time* in the duration of a call. For simplicity we assume that if two predicates $\mathcal{P}_i, P_j$ occur within pattern $\mathcal{S}$ (where $i < j$) and have temporal predicates t_i, t_j, respectively, then these intervals do not overlap and t_i occurs before t_j on the time dimension.

D. Spatio-Temporal Constraints

In order to restrict values that can be matched to spatio-temporal predicates, the STPS language supports an optional set of spatio-temporal constraints $\mathcal{C}$. To qualify a phone user has to first satisfy $\mathcal{S}$ and then $\mathcal{C}$. $\mathcal{C}$ works like a pos-filter to eliminate phone users that do not satisfy $\mathcal{C}$. Some examples of spatio-temporal constraints can be: $@x! = @y$, $@z = \{A, B, C\}$, *Period*(t_i)="Weekend", *Day*(t_i)="Monday", among many others.

E. STPS Language Example

We now provide a complete example of pattern using the STPS language. One example is: "find all mobile users that, on Saturdays, first start in an arbitrary area different to *District-A* in the morning, then immediately went by *downtown*, then by the *Stadium-1* between 6pm and 8pm, then went in the *District-B* neighborhood between 8pm and 10pm, and finally returned to their first area". This query example finds for mobile users that followed a pattern of movements where the first and last locations are not specified but have to be the same ($@x$); three other spatial predicates are defined over areas of interests; several temporal predicates are also defined; and finally spatio-temporal constraints are specified to filter out the results. This pattern query can be expressed in the STPS language as follows: $\mathcal{Q} :=$ ($\langle @x$, t_{from}=6am : t_{to}=12pm$\rangle$. $\langle$*DOWNTOWN*, t_r=1min$\rangle$. $\langle$*STADIUM-1*, t_{from}=6pm : t_{to}=8pm$\rangle$. $\langle$*DISTRICT-B*, t_{from}=8pm : t_{to}=10pm$\rangle$. $\langle @x\rangle$, $\mathcal{C}$=$\{@x$!=*DISTRICT-A*, $\forall t_i, t_j \in \mathcal{S}$, *Date*($t_i$)=*Date*($t_j$) $\wedge$ *Day*(t_i)="Saturday"$\}$).

V. Query Evaluation System

In this section we provide in depth description of the query evaluation system. We start with an overview of the indexing structures used to make the query evaluation more efficient. We then describe the *Index Join Pattern* (*IJP*) algorithm for evaluating pattern queries. This algorithm is based on a *merge-join* operation performed over the *inverted-indexes* corresponding to every fixed predicate in the pattern query $\mathcal{S}$.

A. Index structures

In order to efficiently evaluate pattern queries we use three indexing structures, as shown in Figure 3: (a) one R-tree build on top of the BTS regions; (b) one B$^+$-tree for each BTS$_{id}$ which stores CDR records sorted by *timestamp*; and (c) one *inverted-index* for each BTS$_{id}$ which stores CDR records, sorted first by $phone_{id}$ and then by *timestamp*, that used BTS$_{id}$ sometime during a call. Along with these indexes we also store the CDR records in the archive, grouped by $phone_{id}$ and ordered by *timestamp*, as explained in Section IV. The R-tree is used when there is a spatio-temporal predicate in $\mathcal{S}$ which has some user defined regions (e.g. a spatial range predicate). In this case we have to find the minimal set of coverage areas from the alphabet Σ which completely cover the defined region. In order to do so we create a range query with the user defined region and the R-tree is traversed in order to return the set of BTS that overlap with this region. The records for the returned set of BTS can be merged to form a single list with all entries to be further processed by our algorithm. This is only possible because entries in each *inverted-index* BTS$_{id}$ has its entries ordered by ($phone_{id}$,*timestamp*) key.

The B$^+$-tree is used by the query engine to prune entries that do not satisfy a temporal constraint. The engine makes the decision on using or not the B$^+$-tree based on the type of temporal constraint that is being evaluated (discussed later in this section). The *inverted-index* of a given BTS$_{id}$ stores pointers to all call records that are related to this BTS$_{id}$ in sometime during a call. In the *inverted-index* each entry in BTS$_{id}$ is a record that contains a $phone_{id}$, the *timestamp* and duration during which the user was inside region BTS$_{id}$, and a pointer to the CDR record associated to the call in the CDR archive. If a user connects to a given BTS$_{id}$ multiple times in different *timestamps*, we store a separate record for each use. An example of the indexing structures is shown in Figure 3. The *inverted-index* entry for the region D=231 is {(4333431|1123000|2); (4333432|1021421|3); (4333434|1112141|9); (4333434|1123459|3); ...}. Note that records from an *inverted-index* point to the corresponding phone user in the CDR archive. For example, the record (4333434|1112141|9) in the *inverted-index* 231 contains a pointer to the phone user 4333434.

B. The Index-Join Pattern Algorithm (IJP)

We start with the simple scenario where the pattern $\mathcal{S}$ does not contain any temporal constraints. In this case, the pattern specifies only the order by which its predicates (whether fixed or variable) needs to be satisfied. Assume $\mathcal{Q}$ contains n predicates and let $\mathcal{Q}_f$ denote the subset of f fixed predicates, while $\mathcal{Q}_v$ denotes the subset of v variable predicates (n=f+v). The evaluation of $\mathcal{Q}$ with the proposed algorithm can be divided in two steps: **(i)** the algorithm evaluates the set $\mathcal{Q}_f$ using the *inverted-index* index to fast prune phone users that do not qualify for the answer; **(ii)** then the collection of the reminding *candidate* phone users is further refined by evaluating the set of variable predicates $\mathcal{S}_v$.

(i) Fixed predicate evaluation: All f fixed predicates in $\mathcal{Q}_f$ can be evaluated *concurrently* using an operation similar to a "merge-join" among their *inverted-index* lists $\mathcal{L}_i, i \in 1..f$. Records from these f lists are retrieved in sorted order by ($phone_{id}$,*timestamp*) and then joined by their $phone_{id}$s and *timestamp*. The join criteria is $\mathcal{L}_{i-1}.phone_{id}$ = $\mathcal{L}_i.phone_{id}$ **and** $\mathcal{L}_{i-1}.timestamp < \mathcal{L}_i.timestamp$ (for simplicity we do not consider the dur_i attribute). The first part of the criteria

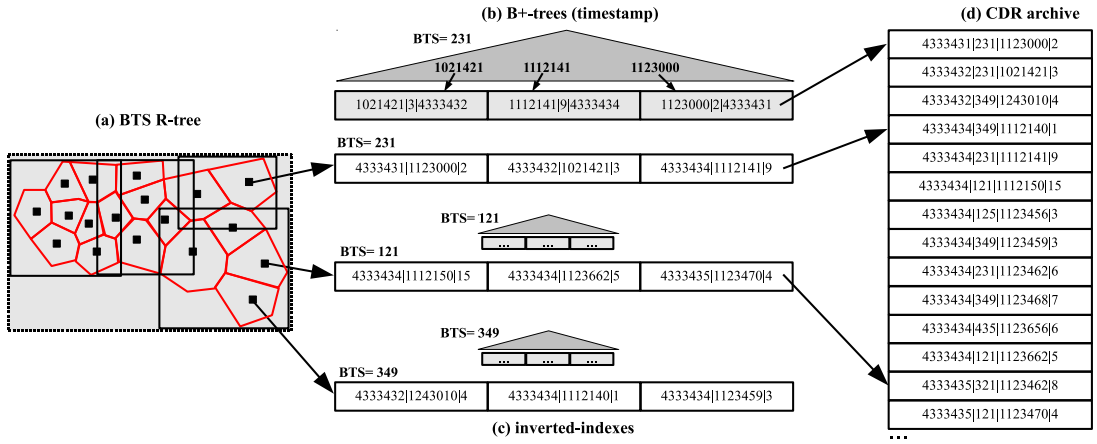

Fig. 3. Index framework: (a) R-tree for the set of BTS; (b) B$^+$-tree and (c) *inverted-index* for each BTS; and (d) CDR archive.

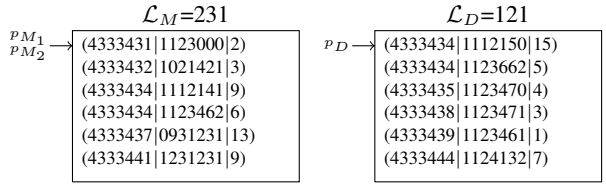

Fig. 4. CDR examples for *inverted-indexes* $\mathcal{L}_M$=231 and $\mathcal{L}_D$=121.

ensures that we are connecting records from the same phone user and the second part ensures that we are satisfying the predicates in the appropriate order. The fact that the records in the *inverted-index* lists are sorted by ($phone_{id}$,*timestamp*) allows us to process the join with a single pass over the lists skipping all records that do not match the join criteria. If the same region appears multiple times in the pattern $\mathcal{S}$ than we use multiple pointers to the *inverted-index* lists for this region.

Example: The first step of *IJP* algorithm is illustrated using the example in Figure 4. Assume the pattern $\mathcal{S}$ in the query $\mathcal{Q}$ contains three fixed and two *variable* predicates, as in: $\mathcal{S} = \{@x.M.D.@x.M\}$. This pattern looks for users that first visited some region denoted by *variable* $@x$, then it visited region M sometime later (no temporal predicate is specified here), then region D and then visited again the same region $@x$ before finally returning to M. The first step of the join algorithm uses the *inverted-index* for M and D ($\mathcal{L}_M$ and $\mathcal{L}_D$). Conceptually, p_{M_1} and p_{M_2} represent two pointers to M *inverted-index* list.

The algorithm starts from the first record in list $\mathcal{L}_M$, phone 4333431, using p_{M_1}. It then checks the first record in list $\mathcal{L}_D$, phone 4333434, using p_D. We can deduce immediately that phone 4333431 is not a candidate since it does not appear in the list of $\mathcal{L}_D$. So we can skip 4333431 and also 4333432 from the $\mathcal{L}_M$ list and continue with the next record, phone 4333434. Since (4333434|1112150|15) in list $\mathcal{L}_D$ has *timestamp* greater than (4333434|1112141|9), these two occurrences of 4333434 coincide with pattern $M.D$ so we need to check if 4333434 uses again region M after *timestamp* 1112150. Thus we consider the first record of list $\mathcal{L}_M$ using

Algorithm 1 *IJP*: Spatial Predicate Evaluation

Require: Query $\mathcal{S}$
Ensure: Phones satisfying fixed $\mathcal{S}_f$ and variable $\mathcal{S}_v$ predicates
1: Candidate Set $U \leftarrow \emptyset$, $f \leftarrow |\mathcal{S}_f|$, $Answer \leftarrow \emptyset$
2: **for** $i \leftarrow 1$ to f **do** ▷ for each $\mathcal{S}_f$
3: Initialize $\mathcal{L}_i$ with the *cell-list* of $\mathcal{P}_i$
4: **for** $w \leftarrow 1$ to $|\mathcal{L}_1|$ **do** ▷ analyze each entry in $\mathcal{L}_1$
5: $p_1 = w$ ▷ set the pointer for $\mathcal{L}_1$
6: **for** $j \leftarrow 2$ to f **do** ▷ examine all other lists
7: **if** $\mathcal{L}_1[w].id \notin \mathcal{L}_j$ **then break** ▷ does not qualify
8: Let k be the first entry for $\mathcal{L}_1[w].id$ in $\mathcal{L}_j$
9: **while** $\mathcal{L}_1[w].id = \mathcal{L}_j[k].id$ **and** $\mathcal{L}_{j-1}[p_{j-1}].t > \mathcal{L}_j[k].t$ **do**
10: $k \leftarrow k + 1$ ▷ align $\mathcal{L}_{j-1}[p_{j-1}].t$ and $\mathcal{L}_j[k].t$
11: **if** $\mathcal{L}_1[w].id \neq \mathcal{L}_j[k].id$ **then break** ▷ does not qualify
12: **else** $p_j = k$ ▷ set the pointer for $\mathcal{L}_j$
13: **if** $\mathcal{L}_1[w]$ qualifies **then**
14: $U \leftarrow U \cup \mathcal{L}_1[w].id$ ▷ $\mathcal{L}_1[w]$ satisfy all $\mathcal{S}_f$
15: **if** $|\mathcal{S}_v| = 0$ **then** $Answer \leftarrow U$
16: **else** ▷ variable predicate evaluation
17: **for** each $u \in U$ **do**
18: $phone_{id} \leftarrow$ Retrieve(u)
19: Build v segments Seg_i using $phone_{id}$
20: Generate *variable-lists* for each segment Seg_i
21: Join *variable-lists*
22: **if** $phone_{id}$ qualifies **then**
23: $Answer \leftarrow Answer \cup phone_{id}$

p_{M_2}, namely user (4333431|1123000|2). Since it is not from 4333434 it cannot be an answer so pointer p_{M_2} advances to record (4333434|1112141|9). Now pointers in all lists point to records of 4333434. However, (4333434|1112141|9) in p_{M_2} does not satisfy the pattern since its *timestamp* should be greater than *timestamp* 1112150 of 4333434 in D. Hence p_{M_2} is advanced to the next record, which happens to be (4333434|1123462|6). Again we have a record from the same user 4333434 in all lists and this occurrence of 4333434 satisfies the temporal ordering, and thus the pattern $\mathcal{S}$. As a result, user 4333434 is kept as a candidate in U. □

In cases where a spatial predicate P_i in $\mathcal{Q}$ is a user defined area, then the above join algorithm has to materialize the *inverted-index* list for the user defined area. This materialized

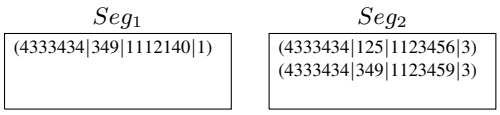

Fig. 5. Segmentation of phone user 4333434 into Seg_1 and Seg_2

list has entries from the set of *inverted-index* lists for the coverage areas in the alphabet Σ which approximate the user defined area. This can be done easily since records in each *inverted-index* list in the coverage area are already ordered by ($phone_{id}$,*timestamp*). Thus, the materialized list can be computed *on-the-fly* by feeding the *IJP* algorithm with the record that has the smallest ($phone_{id}$,*timestamp*) key among the *heads* of the participating *inverted-indexes*.

(ii) Variable predicate evaluation: The second step of the *IJP* algorithm evaluates the v *variable* predicates in $\mathcal{Q}_v$, over the set of candidate phone users U generated in the first step. For a fixed predicate its corresponding *inverted-index* contains all phone users that satisfy it. However, *variable* predicates can be bound to any region, so one would have to look at all *inverted-indexes*, which is not realistic. We will again need one list for each *variable* predicate (termed *variable-list*), however such *variable-lists* are not pre-computed (like the *inverted-indexes*). Rather they are created *on-the-fly* using the candidate phone users filtered from the fixed predicate evaluation step.

To populate a *variable-list* for a *variable* predicate $P_i \in \mathcal{S}_v$ we compute the possible assignments for *variable* P_i by analyzing the *inverted-index* for each candidate phone user. In particular, we use the time intervals in a candidate phone call record to identify which phone call record of the phone user can be assigned to this particular *variable* predicate. An example is shown in Figure 5 using the candidate phone user 4333434. From the previous step we know that 4333434 satisfies the fixed predicates at the following regions: (M,1112141), (D,1112150), (M,1123462). Using the pointers from the *inverted-indexes* of the previous step, we know where the matching regions are in the *inverted-index* of phone user 4333434. As a result, the phone user 4333434 can be conceptually partitioned in two segments: phone call records that happen before p_{M_1}=(4333434|1112141|9) are stored in Seg_1; and phone call records that happen after p_D=(4333434|1112150|15) and before p_{M_2}=(4333434|1123462|6) are stored in Seg_2. Note that records in between p_{M_1} and p_D do not need to be considered.

These segments are used to create the *variable-lists* by identifying the possible assignments for every *variable*. Since a *variable*'s assignments need to maintain the pattern, each *variable* is restricted by the two fixed predicates that appear before and after the *variable* in the pattern. All *variables* between two fixed predicates are first grouped together. Then for every group of *variables* the corresponding segment (the segment between two fixed predicates) is used to generate the *variable-lists* for this group. Grouping is advantageous, since it can create *variable* lists for multiple *variables* through the same pass over the phone user segments. Moreover, it ensures that the *variables* in the group maintain their order consistent with the pattern $\mathcal{S}$.

Assume that a group of *variable* predicates has w members. Each segment that affects the *variables* of this group is then streamed through a window of size w. The first w elements of the phone user segment are placed in the corresponding predicate lists for the *variables*. The first element in the segment is then removed and the window shifts by one position. This proceeds until the end of the segment is reached.

The generated *variable-lists* are then joined in a way similar to the fixed predicate evaluation step. Because the *variable-lists* are populated by records coming from the same user, the join criteria checks only if the ordering of pattern $\mathcal{S}$ is obeyed. In addition, if the pattern contains *variables* with the same name, (e.g. two $@x$ like in our example), the join condition verifies that they are matched to the same region.

1) Temporal Predicate Evaluation: The *IJP* algorithm can easily support explicit temporal predicates by incorporating them as extra conditions in the join evaluations among the list records. There are three cases for a temporal predicate: (1) *interval* time $(t_{from} : t_{to})$; (2) *snapshot* time t_s; or (3) *relative* time t_r.

For the *interval* and *snapshot* temporal predicates, the B$^+$-tree associated to the region in the spatial predicate can be used to retrieve all phone call records that satisfy both spatial and temporal predicates. For the *interval* all records that are within the t_{from} and t_{to}, included, are retrieved, while for the *snapshot* all records that match the t_s temporal predicate are retrieved. Another approach is to verify the *interval* or *snapshot* temporal predicate for each phone call record while processing the *inverted-index* associate to a spatial predicate, without using the B$^+$-tree. In the next section we show that for some types of *interval* temporal predicates, evaluating the *interval* time while processing the *inverted-index* in the *IJP* algorithm is better than accessing the B$^+$-tree index.

For the *relative* time predicate, there are two possible strategies: (1) the straightforward way to evaluate it is, when the spatial predicate is being evaluated, to check whether the temporal predicate is satisfied, in the same way the Algorithm works; (2) another approach is to just use the B$^+$-tree to retrieve all records that satisfy the temporal predicate for P_i when the previous one P_{i-1} was already evaluated. The drawback of this second approach is that, every time a match for P_{i-1} occurs, a search on the B$^+$-tree is performed. If the number of matches for P_{i-1} is high, so the number of searches on the B$^+$-tree, then the first approach become more advantageous. Because the first approach is much simple and seems to be more efficient most of the times, we decided to always perform it when there is a *relative* temporal predicate.

2) Spatio-Temporal Constraints: The evaluation of spatio-temporal constraints $\mathcal{C}$ can be performed as a post filtering step after the pattern $\mathcal{S}$ evaluation is done. Other approaches to verify the set of constraints while processing the spatial predicates are also possible. Due to lack of space we omit further discussions.

VI. Experimental Evaluation

In this paper, we consider two real CDR databases. The first one is a CDR database from an urban environment (hereafter *Urban Database*) and the second one is a CDR database at a state level (*State Database*). The first one is not a subset of the second one. The BTS hopping option was not enabled in either of the databases. The two databases differ regarding both the number of BTSs that the infrastructure has and the spatio-temporal information available for each user (number of calls, frequency of calls, density of BTSs, etc.). This information is to a large extent affected by the sociocultural characteristics of the regions where the data was collected from. Also, these differences deeply affect the number and characteristics of the patterns that can be detected.

Regarding the *Urban Database*, cell phone CDRs for 300,000 anonymous customers from a single carrier for a period of six months were obtained from a metropolitan area. In order to select urban users, all phone calls from a set of BTSs within the city were traced over a 2-week period (sampling period) and the (anonymous) numbers that made or received at least 3 calls per day from those BTSs were selected. Although the selection of subscribers was carried out in an urban environment, they could freely move anywhere within the nation. In total there are around 50,000,000 entries in the database considering voice, SMS and MMS. The BTS database contained the position of 30,000 towers.

As for the *State Database*, we considered 500,000 users from a state for a period of six months. No selection of users was made, i.e. all users that made or received a phone call from any BTS of that particular state during a six month period were included in the database. In total there were close to 30,000,000 entries in the database. The BTS database contained the position of 20,000 towers.

We randomly sampled 500 phone numbers from each database to generate sample queries. For each sampled phone we then randomly selected fragments in its history of calls to generate queries with varying number of predicates. Hence, these queries return at least one entry in their respective databases. For each experiment we measured the average query running time and total number of I/O for 500 queries. The query running time reports the average computational cost (as the total wall-clock time, averaged over a number of executions) for 500 queries. To maintain consistency, we set page size equals to 4KBytes for indexes and data structures. All experiments were performed on a Dual Intel Xeon E5540 2.53GHz running Linux 2.6.22 with 32 GBytes memory.

For evaluation purposes, we compared the *IJP* algorithm against an *extended* version of the *KMP* algorithm proposed in [20], which we call here *Extended-KMP* (*E-KMP*). The *E-KMP* supports **all** spatio-temporal features proposed in our language and process **all** phone users in the CDR database.

A. IJP vs KMP Comparison

In order to preserve details in all graphs, we decided to suppress the *E-KMP* plots since the differences in performance between *E-KMP* and *IJP* are very large. Instead, we describe the results and comparison between both algorithms here in this section. The number of I/O for the *E-KMP* execution is constant in both databases since it performs a sequential scanning of the phone archive. For the *State* database the number of I/O is 1,788,384 per query, while for the *Urban* it is 2,022,020. These values correspond to the total number of data disk pages each database has. The *E-KMP* algorithm performs at least 18 times more I/O than the *IJP* algorithm (for patterns with 2 user defined area predicates with a large number of BTS each for the *Urban* database). This difference is bigger if pattern queries with only spatial predicates are considered. For instance, the difference in total number of I/O for patterns with 4 spatial predicates is 108 and 260 times for the *State* and *Urban* database, respectively.

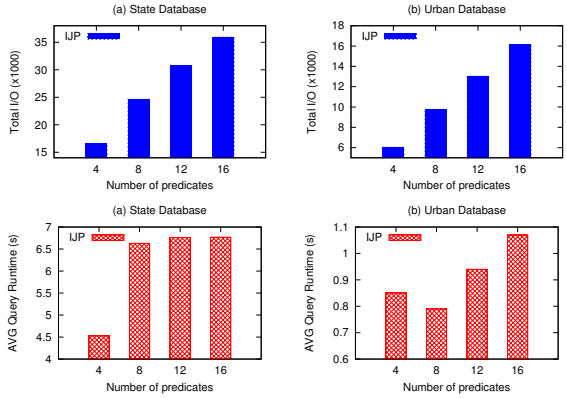

Fig. 6. Total I/O and query runtime for spatial predicates

For the running time the *E-KMP* algorithm on its best performance (patterns with 4 spatial predicates for the *Urban* database) takes on average 853s per query. For the same set of experiment the *IJP* takes on average only 0.85s per query, which makes the *IJP* 1000 times faster than the *E-KMP*. Even though the cost related to I/O operations is constant when increasing the number of predicates for the *E-KMP*, the running time is not. The total time to evaluate patterns with larger number of predicates increases substantially due to the fact that more predicates have to be evaluated for a match.

B. Patterns with Spatial Predicates

The first set of experiments evaluates patterns with different number of spatial predicates. Figure 6 shows the number of I/O (first row) and runtime time (second row) for 4, 8, 12 and 16 spatial predicates. For this kind of patterns only the inverted indexes associated with the predicates in the pattern are accessed. Increasing the number of spatial predicates in the query also increases the number of I/O since more inverted indexes are retrieved. Also, the number of entries to be joined by the *IJP* algorithm increases, which makes the total time increase. On the average 306 and 41 phone users, for the *State* and *Urban* databases respectively, match a pattern.

C. Patterns with Variable Predicates

We also analyzed pattern queries with variables. We tested patterns with 1 variable (Figure 7) and 2 variables (Figure 8),

varying the total number of spatial predicates from 2 to 14. For instance, in the case of patterns with 16 predicates, two query sets were generated: one with 1 variable and 15 spatial predicates; and a second one with 2 variables and 12 spatial predicates. The number of I/O for queries with 4 predicates is bigger than for queries with more predicates for some experiments. This is due to the fact that the CDR database is accessed once a match is found after the *IJP* algorithm evaluates the spatial predicates. This behavior is noticed in all the experiments except for the *Urban* database for patterns with 1 variable.

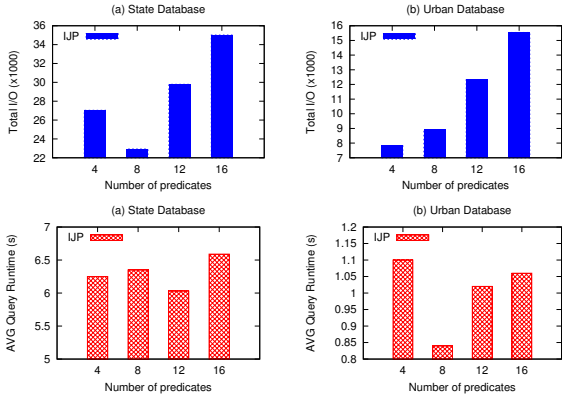

Fig. 7. Total I/O and query runtime for patterns with 1 variable

The differences in the total number of I/O for patterns with 4 predicates increase substantially from 1 to 2 variables. This is due to the fact that the number of spatial predicates drops from 3 to 2, which makes the spatial predicate evaluation phase of the *IJP* algorithm less selective (there are only 2 spatial predicates to filter out CDR entries that for sure do not match the query). Therefore, more CDR entries are analyzed in the variable predicate evaluation phase of the *IJP*. This behavior also occurs, but with small differences, for patterns with 8, 12 and 16 predicates. For these queries the spatial predicate evaluation phase filters out more CDR candidates than queries with only 4 predicates. Therefore, less accesses associated to the phone database are performed, reflecting in the total number of I/O shown in the graphs.

The addition of variable predicates in the pattern also increases the number of matches per query. For instance, for the *Urban* database, on average 41, 230 and 1200 phone users match for patterns with only 4 spatial predicates, 3 spatial and 1 variable predicates, and 2 spatial and 2 variable predicates respectively.

D. Patterns with User Defined Area Predicates

In order to evaluate patterns with user defined area predicates, we generated 1 and 2 user defined area predicates by swapping a spatial predicate with an area containing a set of regions. This set of regions were selected by performing a range query on the BTS locations with center in the original spatial predicate location and a specific window size length. We then swapped the original spatial predicate with the set of

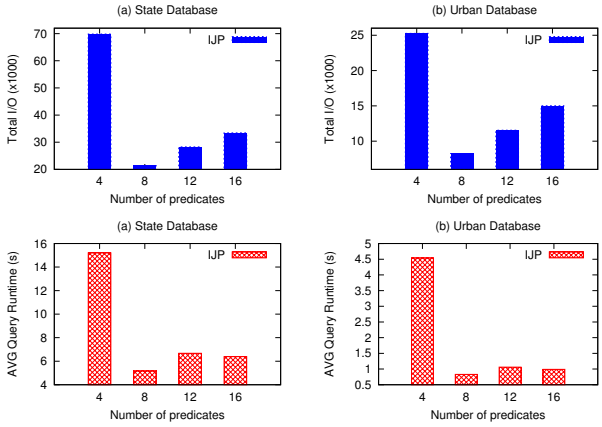

Fig. 8. Total I/O and query runtime for patterns with 2 variables

regions. We generated several query sets for different window size lengths varying from 1 km to 5 km. For the *Urban* database a user defined area predicate contain, on average, 2 regions for 1 Km window size length and 400 regions for 5 Km window size length. For the *State* database the average number of areas selected is up to 130 regions.

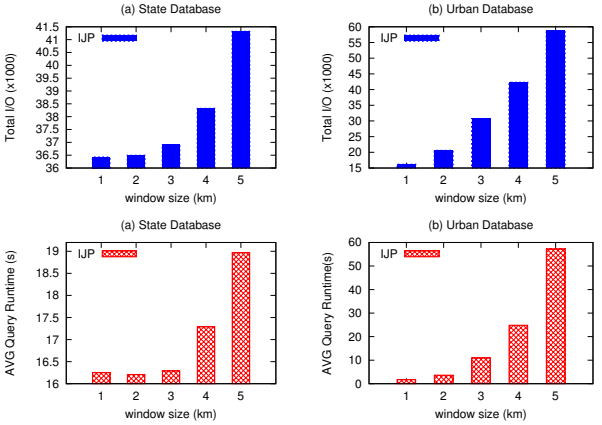

Fig. 9. Total I/O and query runtime for patterns with 1 defined area

Figure 9 and Figure 10 show the results for queries with 1 and 2 user defined area predicates, respectively, for different window size lengths. For large window sizes both the total number of I/O and running time increase because more inverted indexes associated to the user defined area predicates are retrieved. Having many more entries in the inverted indexes also increases the running time since more entries are candidates to be merge-joined by the *IJP* algorithm. The same behavior is noticed when increasing the number of user defined area predicates from 1 to 2.

E. Patterns with Temporal Predicates

In the last set of experiments we evaluated patterns with interval temporal predicates (Figure 11). We generated temporal predicates from the original CDR fragments and then added them to their correspondent spatial predicate. For each pattern query all predicates have two components: a spatial and a

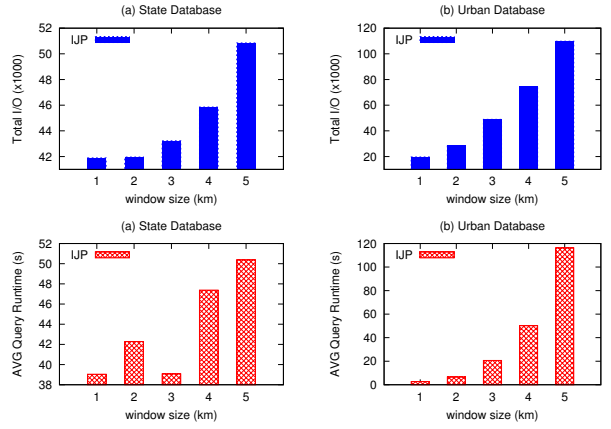

Fig. 10. Total I/O and query runtime for patterns with 2 defined areas

temporal predicate. We then increased the interval in time in all temporal predicates in order to select more candidate entries. The interval values in each temporal predicate range from two days to ten days covering the original timestamp of the CDR database. We evaluated patterns with temporal predicates in two ways (as explained in Section V): the first method (*SEQ*) validates temporal predicates while processing each entry in the inverted indexes; the second method (*INDEX*) employs the B$^+$-tree, for each spatial predicate, to first evaluate the temporal predicate. In *INDEX*, entries that satisfy the temporal predicate are further grouped by $phone_{id}$ and then sorted by *timestamp* to be further processed by the *IJP* algorithm.

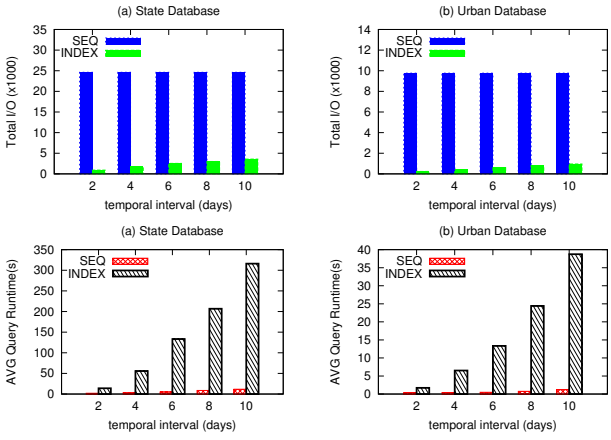

Fig. 11. Total I/O and query runtime for patterns with temporal predicates

The total number of I/O for the *SEQ* method is constant since, for each spatial predicate, all pages in the inverted indexes are accessed. On the other hand, the number of I/O for the *INDEX* approach is much smaller than *SEQ* since only entries that satisfy the temporal predicates are retrieved. The running time of the *INDEX* approach is worse than in the *SEQ* method when increasing the interval time. This is due the factor that many more entries retrieved by the B$^+$-tree need to be further sorted before being analyzed by the *IJP* algorithm. The *INDEX* approach start to become more appealing for temporal predicates with high selectivity (e.g. temporal predicates with interval less than 1 hour (not shown in the graphs)).

VII. Conclusions and Future Work

The ability to detect and characterize mobility patterns using CDR databases opens the door to a wide range of applications ranging from urban planning to crime or virus spread. Nevertheless, the spatio-temporal query systems proposed so far cannot express the flexibility that such applications require. In this paper we described the Spatio-Temporal Pattern System (STPS) for processing spatio-temporal pattern queries over mobile phone-call databases. STPS defines a language to express pattern queries which combine fixed and variable spatial predicates with explicit and implicit temporal constraints. We described the STPS index structures and algorithm in order to efficiently process such pattern queries. The experimental evaluation shows that the STPS can answer spatio-temporal patterns very efficiently even for very large mobile phone-call databases. Among the advantages of the STPS is that it can be easily integrated in commercial telecommunication databases and also be implemented in any current commercially available RDBMS. As a next step we are extending the STPS to evaluate continuous pattern queries for streaming phone-call data.

References

[1] K. Dasgupta and et al., "Social ties and their relevance to churn in mobile telecom networks," in *EDBT*, 2008, pp. 668–677.

[2] A. Nanavati and et al., "On the structural properties of massive telecom call graphs: findings and implications," in *ACM CIKM*, 2006.

[3] M. Seshadri and et al., "Mobile call graphs: beyond power-law and lognormal distributions," in *ACM SIGKDD*, 2008, pp. 596–604.

[4] E. Halepovic and C. Williamson, "Characterizing and modeling user mobility in a cellular data network," in *ACM PE-WASUN*, 2005.

[5] M. C. Gonzalez, C. A. Hidalgo, and A.-L. Barabasi, "Understanding individual human mobility patterns," *Nature*, no. 7196, June 2008.

[6] H. Zang and J. Bolot, "Mining call and mobility data to improve paging efficiency in cellular networks," in *ACM MobiCom'07*, 2007.

[7] D. Knuth, J. Morris, and V. Pratt, "Fast pattern matching in strings." *SIAM J. on Computing*, 1977.

[8] M. Vieira, P. Bakalov, and V. Tsotras, "Querying trajectories using flexible patterns," in *EDBT*, 2010.

[9] R. Sadri and et al., "Expressing and optimizing sequence queries in database systems," *ACM TODS*, 2004.

[10] P. Seshadri, M. Livny, and R. Ramakrishnan, "SEQ: A model for sequence databases," in *IEEE ICDE*, 1995.

[11] J. Agrawal and et al., "Efficient pattern matching over event streams," *SIGMOD*, pp. 147–159, 2008.

[12] M. Erwig and M. Schneider, "Spatio-temporal predicates," *TKDE*, 2002.

[13] M. Hadjieleftheriou, G. Kollios, P. Bakalov, and V. Tsotras, "Complex spatio-temporal pattern queries," in *VLDB*, 2005, pp. 877–888.

[14] A. Anagnostopoulos and et al., "Global distance-based segmentation of trajectories," in *ACM KDD*, 2006.

[15] Y. Cai and R. Ng, "Indexing spatio-temporal trajectories with Chebyshev polynomials," in *ACM SIGMOD*, 2004.

[16] M. Vlachos, G. Kollios, and D. Gunopulos, "Discovering similar multidimensional trajectories," in *IEEE ICDE*, 2002.

[17] D. Pfoser, C. Jensen, and Y. Theodoridis, "Novel approaches in query processing for moving object trajectories," in *VLDB*, 2000.

[18] M. Hadjieleftheriou, G. Kollios, V. Tsotras, and D. Gunopulos, "Indexing spatiotemporal archives," *VLDB J.*, pp. 143–164, 2006.

[19] Y. Tao and D. Papadias, "MV3R-Tree: A spatio-temporal access method for timestamp and interval queries," in *VLDB*, 2001, pp. 431–440.

[20] C. du Mouza, P. Rigaux, and M. Scholl, "Efficient evaluation of parameterized pattern queries," in *CIKM*, 2005, pp. 728–735.