Querying Geo-social Data by Bridging Spatial Networks and Social Networks

Yerach Doytsher
Technion
Haifa, Israel
doytsher@technion.ac.il

Ben Galon[*]
Technion
Haifa, Israel
bgalon@technion.ac.il

Yaron Kanza[*]
Technion
Haifa, Israel
kanza@cs.technion.ac.il

ABSTRACT

Recording the location of people using location-acquisition technologies, such as GPS, allows generating *life patterns*, which associate people to places they frequently visit. Considering life patterns as edges that connect users of a social network to geographical entities on a spatial network, enriches the social network, providing an integrated socio-spatial graph. Queries over such graph extract information on users, in correspondence with their location history, and extract information on geographical entities in correspondence with users who frequently visit these entities.

In this paper we present the concept of a socio-spatial graph that is based on life patterns, where users are connected to geographical entities using life-pattern edges. We provide a set of operators that form a query language suitable for the integrated data. We consider two implementations of a socio-spatial graph storage—one implementation uses a relational database system as the underline data storage, and the other employs a graph database system. The two implementations are compared, experimentally, for various queries and data. An important contribution of this work is in illustrating the usefulness and the feasibility of maintaining and querying integrated socio-spatial graphs.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*

General Terms

Experimentation, Performance

Keywords

Geographic information systems, social networks, Spatio-social applications, geosocial networking, spatial query language, graph database

[*]The work of these authors was partially supported by the Israeli Ministry of Science and Technology, Grant 3/6472.

1. INTRODUCTION

Location-acquisition technologies allow tracing the location history of individuals. Such data can be collected and analyzed to discover frequently visited locations. For instance, when mobile phones are connected to a GSM (Global System for Mobile Communications), positioning that discloses the actual coordinates of the mobile phone can be stored in a log along with the time stamps of the positioning events. In other scenarios, users may activate a GPS-based location-tracing application and record their location history as a sequence of location-time events.

Given a log of location-time events, the data can be analyzed to extract *life patterns*. Essentially, a life pattern synopsizes the frequency of a visit in a certain location or in a geographical entity, where both the frequency and the location can be at different granularities. For example, a life pattern P of a user Alice may specify that Alice visits Westminster Cathedral every Sunday. Another life pattern may specify that Alice stays in 49 Piccadilly every workday from 9 AM to 5 PM.

However, in real life users do not tend to strictly act according to behavioral patterns. Thus, it should be possible that patterns will reflect behaviors whose regularity is somewhat flexible. In such cases, we specify the level of flexibility by a *confidence* value that is attached to each pattern. Intuitively, the confidence value of a pattern indicates the likelihood of the pattern to hold in the times it relates to. For instance, suppose that when analyzing the recordings of the locations of Alice it is revealed that Alice visited Westminster Cathedral only in 80 Sundays out of the 100 Sundays on which her locations were recorded. In such case, the confidence of the pattern P, mentioned above, is 0.8.

Extracting life patters of users from location loggings has been studied by Ye *et al.* [8]. Thus, we assume in this paper that their techniques are being employed for the generation of the life patterns.

Life patterns can enrich a *social network* by connecting it to a *spatial network*. A social network is essentially a large graph where the nodes represent users and the edges represent relationships between users. The spatial network is essentially a large graph where the nodes represent geographical entities and the edges represent roads that connect adjacent entities. A *joined socio-spatial graph* is created by combining a social network with a spatial network, connecting their nodes by *life-pattern edges*, where each user is connected to the places she frequently visits.

In order to exploit joined socio-spatial graphs, it should be possible to easily and efficiently query the data. A suitable query language should be designed to cope with the structure of the data being a graph, and it should be adapted to graphs in which the nodes are divided into groups—some nodes represent users while other nodes represent geographical entities. Moreover, there are three types of edges in such graphs—edges that represent friendship relationships between users, edges that represent adjacency between geographical entities and life-pattern edges that represent frequent visits of users in geographical entities. The query language should be able to discern between them and use the appropriate edges according to the query.

In this paper we present a set of operators that serve as the building blocks of a socio-spatial query language over a joined socio-spatial graph. We illustrate the effectiveness and the expressiveness of these operators, to extract valuable information from such graph. Finally, we describe two implementations of a system that provides a storage of a socio-spatial graph and supports the operators. In one implementation, a relational database system serves as the underline data storage. In the other implementation, a graph database is being used. Our experiments demonstrate the efficiency of the two implementations for different operators.

The paper is organized as follows. In Section 2 we formally present our framework. The operators of the query language are presented in Section 3. Two implementations of our approach are described in Section 4. In Section 5 we provide an experimental comparison of the two implementations. Finally, in Section 6 we discuss related work and in Section 7 we conclude.

2. FRAMEWORK

In this section we present our framework and provide the formal definitions of our model. We formally define two types of networks—social network and spatial network—and we define the *joint spatio-social graph* that combines the two networks by connecting their nodes using life-pattern edges.

Social Network. A social network is a graph whose nodes (also called *users*) represent real-world people and whose edges represent relationships (typically, friendship relationships) between people. Each user has attributes specifying personal properties of the person it represents. Name and hobbies are examples of personal properties.

Formally, a social network is an undirected graph $N_{social} = (U, F)$, where U is a set of nodes (users) and $F \subseteq U \times U$ is a set of edges. An edge $(u_1, u_2) \in F$ is called a *friendship edge* between u_1 and u_2.

Geographical Hierarchy. In this paper we consider a setting where geographical entities at different scales are mixed. Consequently, a geographical entity may contain another geographical entity. For example, a city contains its neighborhoods and a neighborhood contains its buildings.

Each geographical entity has an area that is represented as a polygonal shape. Given a set E of geographical entities, we say that for two entities $e, e' \in E$ holds that e is *contained* in e' if the area of e is contained in the area of e'. We say that e is a *sub-entity* of e' if (1) e is contained in e', and (2) there is no entity $x \in E$ such that e is contained in x and x is contained in e'.

A *geographical hierarchy* with respect to a set E of geographical entities is a directed graph $G_H = (E, H)$ where $H = \{(e, e') \mid e \textit{ is a sub-entity of } e'\}$, i.e., there is an edge in H for every pair of an entity and its sub-entity. We say that two entities e_1 and e_2, in E, are *adjacent*, if (1) both e_1 and e_2 are siblings in G_H, i.e., there is $e'' \in E$ such that both e_1 and e_2 are sub-entities of e''; and (2) there is a road that connects e_1 and e_2, and this road does not go via any other of their siblings.[1]

Spatial Network. A *spatial network* is a graph whose nodes represent geographic entities and whose edges represent roads that connect adjacent entities.

Formally, a spatial network is a graph $N_{spatial} = (O, R)$, where O is a collection of geo-spatial objects, representing real-world geographical entities, and $R \subseteq O \times O$ is a set of edges, connecting objects that represent adjacent entities. Objects of O have spatial attributes such as location or shape, and they may have non-spatial attributes such as name or address.

Time Hierarchy. A time hierarchy is a tree whose nodes are time units (e.g., week), and whose edges represent containment relationships between the time units. We denote a time hierarchy as (T, H_T), where T is a set of time units and H_T is a tree over T, defining a hierarchy. For example, in the hierarchy there may be a year as a parent of month, month as a parent of week, week as a parent of day. A *time pattern* is being used to represent a list of days that specify a part of a time unit (e.g., workdays or Sundays are parts of a week), and hours that specify the relevant part of the day. A time pattern has the form $TP = (tr, [d_1, \ldots, d_k], st, et)$ where tr is a time unit being referred to as the *time reference* of the pattern (e.g., year, month, week, day), $[d_1, \ldots, d_k]$ is an array of integers specifying the relevant days with respect to the time reference, st and et are start and end times, in hours, during the day. For example, (*week*, $[2, 3, 4, 5, 6]$, 9, 17) is a patterns that specifies the hours 9AM to 5PM in the workdays Monday to Friday, (*day*, [1], 22, 5) represents nighttime, and (*month*, [1], 10, 14) represents the hours 10AM to 2PM in the first day of the month. Typically, we will use aliases for time patters, e.g., 'Sundays' for (*week*, [1], 1, 24).

We can use the time hierarchy and the time patterns to infer visit patterns. If a certain event often occurs, at a frequency that refers to a certain time unit, then it also occurs with respect to time units that are higher in the hierarchy. For instance, if Alice visits Westminster Cathedral every Sunday then she visits the Cathedral every week and also does so every month (because the 'Sundays' time pattern has a time reference of a week and a week appears below month in the hierarchy).

Life Pattern. Associations between users and geographic entities are expressed by *life patterns.* A life pattern specifies that a certain individual visits a certain geographical entity at a specified frequency.

Consider a geographical hierarchy $G = (E, H_G)$ and a time hierarchy (T, H_T). Given a set of users U, a life pattern is a 4-tuple $P = (u, e, t, c)$, where $u \in U$ is the user of P, $e \in E$ is the geographical entity of P, t is the time pattern of P, and $0 \leq c \leq 1$ is the *confidence* of P. Intuitively, the confidence c is according to the percentage of cases where the log of positioning events supported the life pattern. To explain that more precisely, we briefly describe the generation of life patterns.

[1]A different definition of adjacency can be used here without affecting the rest of the paper, however, this definition is intuitive for the operators that we will present in Section 3 and it simplifies the implementation of our approach.

For generating life patterns, initially the signals of a positioning device are identified and stored. We refer to each such signal as a *sigprint*. Each sigprint contains the identity of the user, the location and the time of the positioning event. A life pattern is generated from a set of sigprints that are all related to the same user, and all of them are located in the area of the same geographic entity. A life pattern is generated for a given user u and geographic entity e. We consider the set $\mathit{sigprint}_u(e)$ to be the set of all sigprints that are associated to the user u and that were measured in a location inside the area of e. Then, given a time pattern t, we divide the measuring period to time slots, according to the time reference of t, and we consider c as the percentage of slots for which there exists a sigprint in $\mathit{sigprint}_u(e)$. For instance, if u is Alice, e is Westminster Cathedral, t is a week, and the data comprises sigprints that were recorded in a time period of 50 weeks, then c is the percentage of weeks (during the time period where sigprints were recorded) for which there is a sigprint that positions Alice in Westminster Cathedral, i.e., in how many weeks, out of the 50 relevant weeks, Alice visited the Cathedral.

Typically, only life patterns whose confidence exceeds a given threshold are of interest. In contrast, life patterns should be as specific as possible. For instance, it may be possible to connect Alice both to Westminster Cathedral and to London, for the Sundays time pattern, because Westminster Cathedral is in London. In such case, if the frequency of visits of Alice in Westminster Cathedral exceeds the required confidence value, we add a life-pattern edge from Alice to Westminster Cathedral, and not to London, and we assume that the relationship to London can be inferred from the geographical hierarchy. Finding the most specific life patterns, for a given threshold, can be done naively by considering for each user all the pairs of geographical entity and time pattern. More efficient methods employ techniques that are similar to those being used in data mining. For details, see the work of Ye *et al.* [8].

Spatio-social Network (SSN). A *joint spatio-social network* (SSN) is a graph that integrates a social network and a spatial network by connecting the users to geographic locations using life-pattern edges.

Let $N_{social} = (U, F)$ be a social network and $N_{spatial} = (O, R)$ be a spatial network. Consider a geographical hierarchy $G = (O, H_G)$, a time hierarchy (T, H_T) and a confidence threshold $0 \leq \tau \leq 1$. Then the joint spatio-social network of N_{social} and $N_{spatial}$ is a graph $N_{join} = ((U, F), (O, R), B)$, where (U, F) is the given social network, (O, R) is the given spatial network, and $B \subset U \times O \times T \times [\tau, 1]$ is a set of life patterns for the users U with respect to the geographic objects O and the time units T. Note that only life patterns whose confidence exceeds the threshold τ can be used for the connection. We refer to the set B of life patterns as the *bridge* between N_{social} and $N_{spatial}$.

A 4-tuple of a user, geographic object, time pattern and confidence value can serve as a constraint, and accordingly, can be used to select life-pattern edges, in the following way. We say that a life-pattern edge (u, o, t, c) satisfies a 4-tuple (u', o', t', c'), considered as a *bridge constraint*, if (1) $u' = u$ (i.e., it is the same user), (2) $o' = o$ or o' is an ancestor of o in the geographical hierarchy G, (3) t' satisfies t according to the time hierarchy (T, H_T), and (4) $c' \geq c$. For example, ('John Smith', 'Westminster Cathedral', Sundays, 0.7) satisfies the bridge constraint ('John Smith', 'London', once a week, 0.7). That is, if our recordings show that John visits Westminster Cathedral in at least 7 out of 10 Sundays, then we know he is in London in at least 7 out of 10 weeks. This is because Westminster Cathedral is in London (according to the geographical hierarchy) and 'every Sunday' is a time pattern that satisfies 'every week' (according to the time hierarchy).

3. QUERY LANGUAGE

To easily and effectively query an integrated socio-spatial network, a suitable query language should be used. In this section we present a set of operators that are the building blocks of such language, and we provide examples that illustrate the effectiveness of the proposed operators.

Throughout this section we assume that the SSN has the form $N_{join} = (N_{social}, N_{spatial}, B)$, and we use the notations of the previous section, i.e., $N_{join} = ((U, F), (O, R), B)$.

3.1 Operators

The query language has the form of an algebra with six operators (seven, if considering bridge and multi-bridge as two different operators). We refer to it by the name SSNA (Socio-Spatial Network Algebra).

Select. The *select operator* receives a set of nodes and a condition, and it returns all the nodes, in the set, that satisfy the condition. It may receive a network, instead of a set, and then the condition is applied over all the nodes of the network. It is written `Select(`S`, `C`)`, where $S \subseteq U$ or $S \subseteq O$ is a set of nodes of some network, and C is a condition. The condition C is with respect to the attributes of the nodes, and it may include `and`, `or`, `not` and `like`, with syntax and semantics similar to those of SQL. For example, in a social network N_{social} where users have an address attribute, the following query

`Select(`N_{social}`, address like '%Downing Street%')`

will return all the people in the social network whose address is in Downing Street.

Extend. The *extend operator* receives a set of nodes, of some network, and a length limit. It returns all the nodes that are reachable by a path whose length does not exceed the given limit (including the nodes of the given set). We write it as `Extend(`S`, `n`)`, where $S \subseteq U$ or $S \subseteq O$ is the set of nodes and n is the length limit.

Formally, a path in a graph $G = (V, E)$ is a sequence of nodes $v_1, \ldots, v_k$, in V such that every two adjacent nodes in the sequence are connected by an edge in G (i.e., for every v_i, v_{i+1} where $1 \leq i \leq k-1$, exists an edge $(v_i, v_{i+1}) \in E$). The length of a path with k nodes is $k-1$ (because we count the number of edges that connect adjacent nodes). We say that the length from node v' to node v'' is k, denoted $\mathit{length}(v', v'') = k$, if the length of the shortest path in G from v' to v'' is k, and for each node v, $\mathit{length}(v, v) = 0$. Now, given a set of nodes $S \subseteq V$ in G, `Extend`$(S, n) = \{v \mid \exists v' \in G\ (\mathit{length}(v, v') \leq n)\}$.

For example, consider the following query over N_{social}:

`Extend(Select(`N_{social}`, address like '%10 Downing Street%'), 2)`

It will returns people who live in *10 Downing Street*, people who are connected in the social network to those who live in

10 Downing Street, and those who are connected by a path of length two to people who live in *10 Downing Street.*

Union, Intersect, Difference. The set operators *union, intersect* and *difference* are applied over sets of nodes in an ordinary fashion, under set semantics, i.e., without repetitions of nodes.
`Union(`S_1`, `S_2`)` $= \{v \mid v \in S_1 \textit{ or } v \in S_2\}$;
`Intersect(`S_1`, `S_2`)` $= \{v \mid v \in S_1 \textit{ and } v \in S_2\}$;
`Difference(`S_1`, `S_2`)` $= \{v \mid v \in S_1 \textit{ and } v \notin S_2\}$.

For example, the following query over N_{social}

```
Difference(Select(N_social, occupation='Politician'),
Extend(Select(N_social, address like '%10 Downing
Street%'), 1))
```

returns politicians who are not directly connected, according to N_{social}, to people who live in *10 Downing Street.*

Bridge. In the operators we presented so far, the input set and the output set belong to the same network. The *bridge* operation is a novel operation that receives nodes of one network and returns nodes of the other network. It does so by moving from one set of nodes, of one network, to the set of nodes, of the other network, that are connected to the nodes of the first set by life-pattern edges.

We denote the bridge operator by `Bridge(`S`, `TP`, `c`)`, where S is a set of nodes such that $S \subseteq U$ or $S \subseteq O$, TP is a time pattern, and c specifies the minimal confidence of the life patterns being used for the bridging.

For a set $S \subseteq U$ of users in N_{social}, the result of the operation `Bridge(`S`, `TP`, `c`)` is $\{o \in O \mid \exists (u,o,t,c') \in B\ (u \in S \textit{ and } (t \textit{ satisfies } TP) \textit{ and } c' \geq c\}$. That is, the result consists of the geographical objects of $N_{spatial}$ that are connected by a life-pattern edge $b \in B$, to a node of S such that the time pattern TP is satisfied and such that the confidence of b is not less than c.

Similarly, for a set $S \subseteq O$ of geographic objects in $N_{spatial}$, the result of the operation `Bridge(`S`, `TP`, `c`)` is $\{u \in U \mid \exists (u,o,t,c') \in B\ (o \in S \textit{ and } (t \textit{ satisfies } TP) \textit{ and } c' \geq c\}$. That is, the result consists of the users of N_{social} that are connected by a life-pattern edge $b \in B$, to a node of S such that the time pattern TP is satisfied and such that the confidence of b is not less than c.

For example, the following query over N_{social}

```
Bridge(Select(N_social, address like '%10 Downing
Street%'), 'every week', 0.8)
```

returns the places (geographic entities) that people who live in *10 Downing Street* visit in 80% of the weeks. Here `'every week'` is an alias of a time pattern that specifies a visit per week, i.e., (*week*, [1,2,3,4,5,6,7], 1, 24).

The previous example applied a bridge from the social network to the geographical network. The following example shows how a bridge can be applied from the geographical network to the social network.

```
Bridge(Select(N_spatial, address like '%10 Downing
Street%'), 'every week', 0.8)
```

It returns the people who visit *10 Downing Street* almost every week (i.e., every week with confidence 0.8).

We can also apply the bridge operation back and forth. For example,

```
Joggers = Select(N_social, hobby='Jogging')
Parks = Select(Bridge(Joggers, 'every day', 0.8),
type='park')
LikelyJoggers = Bridge(Parks, 'every week', 0.9)
```

will find people who almost every week visit a park that people whose hobby is jogging visit almost every day. Note that to simplify the presentation of the expression, we divided it into subexpressions and assigned the subexpressions to node-set variables, e.g., the variable `Joggers`. We used a variable in expressions that follow the assignment to it. An unfolding, where each variable is replaced by the expression it represents, recursively, can eliminate the variables and produce a single expression that answers the query.

Consider two objects $o \in O$ and $u \in U$. Because the edges of B are undirected, we have the following symmetry. The object o satisfies $o \in$`Bridge(`$\{u\}$`, `TP`, `c`)` if and only if u satisfies $u \in$`Bridge(`$\{o\}$`, `TP`, `c`)`. In such case we say that o is (TP, c)-*bridged* to u, and u is (TP, c)-bridged to o.

Multi Bridge. In the bridge operation, a node is in the result if it is connected to a node of the other network by an edge that satisfies the time pattern and the confidence requirement. The *multi-bridge* operator generalizes the bridge operation by requiring from a node in the result to be connected to a certain percentage of the nodes in the given set.

We denote the operation by `MBridge(`S`, `TP`, `c`, `per`)`, where S is the given set of nodes ($S \subseteq O$ or $S \subseteq U$), TP is the specified time pattern, c is the required confidence and *per* is the percentage of objects of S that an object in the result should be connected to. That is, given $S \subseteq U$, the answer to `MBridge(`S`, `TP`, `c`, `per`)` consists of all the objects o such that for at least *per* percentage of the objects $u \in S$, o is (TP, c)-bridged to u. Similarly, when $S \subseteq O$, the answer is the set of users u such that the percentage of objects of S that u is (TP, c)-bridged to is at least *per*.

The multi bridge operation can be used to discover groups of people who have socio-spatial similarity in the sense that they all visit similar locations. It can, as well, be used to discover social similarity among locations in the sense that two locations are *socially similar* when many people who frequently visit the first location also frequently visit the second location, and vice versa.

For example, suppose we want to search for potential new friends for John Smith. We start by finding his friends.

```
FriendsOfJohn = Extend(Select(N_social, name='John
Smith'), 1)
```

Then, we use the multi-bridge operator to find the geographical entities that are defined as 'entertainment' sites and that many of John's friends (60 percent of them) visit nearly every week.

```
Entertainment = Select(MBridge(FriendsOfJohn,
'every week', 0.8, 60%), category='entertainment')
```

Finally, we find people who frequently visit 80% of the entertainment venues that John's friends frequently visit.

```
PotentialNewFriends = MBridge(Entertainment,
'every week', 0.8, 80%)
```

This example illustrates the use of the multi-bridge operator as a way to find socio-spatial similarity among people. Finding social similarity among geographical entities can be done in the same fashion. For instance, we can find locations that are defined as theaters, discover the people who frequently visit a theater, and find other types of places that many theater lovers frequently visit.

```
Theaters = Select($N_{spatial}$, type=‘theater’)
T-Lovers = Bridge(Theaters, ‘every month’, 0.8)
SimilarToTheater = MBridge(T-Lovers, ‘every
month’, 0.8, 60%)
```

3.2 Comparison to Relational Algebra

SSNA has some similarity to relational algebra. Mainly, the `select`, `union`, `set difference` and `intersect` operators are shared by both algebras. However, there are inherent differences between the two languages.

First, expressing the `extend` and `bridge` operations in relational algebra is cumbersome because algebra is not designed for graph traversal. Moreover, expressing the multi bridge operation requires the use of aggregation functions which are not in the core of relational algebra.

Second, SSNA does not include Cartesian product and projection. The essence of the algebra is to have in the result of each expression a set of nodes and not a set of tuples. Thus, there is no need for Cartesian product to generate tuples. Similarly, using projection to disassemble tuples is superfluous.

Dealing merely with sets of nodes, and not with tuples, reduces the expressiveness of the algebra, however, it prevents the exponential blowup that can be caused by a Cartesian product. This is necessary to handle large networks.

Third, relational algebra is not designed for using a time hierarchy and a geographical hierarchy to infer satisfaction of time patterns and of bridge constraints. In SSNA such inference is performed transparently.

4. IMPLEMENTATION

We implemented the proposed model and SSNA query language in two different manners. One implementation uses a graph database as the underlying data storage. The other implementation uses a relational database to store the data. Our goals were to experimentally compare these two approaches, to demonstrate the feasibility of the model, and to show that a socio-spatial network can be built effectively upon common data-storage tools.

4.1 Building on a Graph Database

A *graph database management system* is a system that is designed to store and manage data whose model is a graph. Consequently, such system provides a natural storage for socio-spatial networks. The system provides an interface to store the nodes of the graphs, along with their attributes, and storing labeled edges, either as directed or as undirected edges. In our setting, the labeled edges are needed to store life-pattern edges where the time patterns and the confidence values are encoded in the labels.

In our implementation, we used Neo4j [7]—an open source graph database management system, written in Java. It is a transactional system that provides persistence by managing data on the disk. An indexing mechanism is used for efficient retrieval of the nodes.

The spatial network is stored as a graph whose nodes have different attributes (e.g., address and type). Appropriate indexes where created for these attributes. The time hierarchy and the location hierarchy are stored as tree graphs. The social network was also stored as a graph whose nodes have attributes (e.g., name, occupation and hobbies) and are indexed. The edges are the friendship relationships.

4.2 Building on a Relational Database

The graph database management system, we discussed earlier, provides a natural data storage for socio-spatial networks, because it was designed to store networks. In comparison, when using a relational database-management system to store the data, a model translation should be applied, exchanging graphs for relations.

In the model translation, for each basic network, two or three relations are created. One relation is created to store the nodes, and one or two relations to store the edges.

For the spatial network, one table stores the geographic objects, a second table stores the hierarchical relationship, and a third table stores the adjacency relationship. An identifier is created for the objects, and indexes are constructed for these identifier columns, in order to efficiently retrieve the edges of each node, and the nodes of each edge.

Each row in the adjacency-relationship table represents an edge in the spatial network. For reducing disk space, one may store each pair of adjacent objects o_1 and o_2 as a single tuple and write the queries that retrieve data as a union of two queries, where each query searches in a different column. For example, in a search for the neighbors of o_2, one query will return the nodes o' that have a tuple (o_2, o') in the table, the other query will return the nodes o'' that have a tuple (o'', o_2) in the table, and the answers to these queries will be combined using `union`. The problem with this approach is that it is not designed for efficient use of indexes and that `union` is an expensive operation.

Thus, we chose a different approach. We store for each pair of adjacent objects o_1 and o_2 the two tuples (o_1, o_2) and (o_2, o_1). The cost of maintaining such table is a bit higher that the approach above, however, retrieval of neighbor objects is efficient when using an index on the first column.

Storing the geographic hierarchy raised a difficulty similar to the one described for the adjacency-relationship table. If the storage is simply a collection of parent-child pairs, finding ancestors or descendants of a node is inefficient in SQL because many joins are involved. Furthermore, if the hight of the hierarchy is unknown, recursion should be used, so SQL cannot handle such tasks. Our solution to this was, again, adding redundant edges to the tables—scarifying management efficiency (i.e., the efficiency of insertions, updates and deletions) for efficient retrieval. We stored for each pair o_a and o_d, of ancestor and descendant objects, a 3-tuple (o_a, o_d, d), where d is the distance between the nodes. For instance, if o_a is a parent of o_d then d is 1. If o_a is a parent of a parent of o_d then d is 2, and so on. Because the hierarchy is a tree, at the worst case, we store each object h times, where h is the height of the hierarchy.[2]

The storage of the social network is similar to that of the spatial network, using two tables—one table to store the nodes, and a second table to store the friendship edges, with the redundancy described above. The time hierarchy is stored in a table using the same approach as the one we used to store the geographic hierarchy.

[2]In practice, h is expected to be about 4, and in typical scenarios, the increase in the size of the storage worths the improvement in the efficiency of query evaluation.

Our implementation was built on top of the MySQL relational database management system [5], using the InnoDB transactional storage engine. MySQL is an open source database management system that employs state-of-the-art storage and indexing techniques.

4.3 Implementing SSNA

We implemented the socio-spatial network algebra over both the graph database and the relational database. We now briefly discuss the implementation of the operators over the different databases.

For the graph database, `Select` was implemented as a direct retrieval of nodes using the index. `Extend` was implemented as a BFS (breadth-first search) traversal. `Bridge` was implemented as a scan over all the bridge edges and their labels, checking whether the bridge condition is satisfied (while considering the hierarchies of time and location). The multi-bridge operation is based on the bridge operation. It employs the bridge operation, aggregates the results and checks satisfaction of the multi bridge condition, according to the specified percentage. `Union`, `Intersect`, and `Difference` are implemented using standard methods Neo4j provides. These methods are similar to the methods of the package `java.util.List`.

In the relational model, each SSNA expression is translated to an SQL query and the SQL query is evaluated over the tables of the database. Initially, each operator is written as an SQL query, and the operators are combined by nesting them in the FROM clause.

For example, consider the following query:

`Select(`N_{join}`, name='John Smith')`

The system translates it to the following SQL query.

```
Select user_id
From Friends
Where name = 'John Smith';
```

If the above query is embedded in another expression then it will appear in the FROM clause of the embedding generated SQL query. For example, for the following expression

`Extend(Select(`N_{join}`, name='John Smith'),2)`

the system translates it to the following SQL query.

```
Select f2.friend_id
From Friends f2, (Select user_id
                  From Friends
                  Where name = 'John Smith') f1
Where f1.friend_id=f2.user_id;
```

Note that for `Extends(..., 2)` the answer should comprise nodes reachable by paths of length 0, 1 and 2. This can be computed as a union of three different queries. Our solution is simpler and more efficient. We include in the Friends table self loops, i.e., a tuple (u, u) for each user u. It is easy to see how self loops solve the problem efficiently. Since the translation of the SSNA operators to SQL is technical, we do not elaborate on that further.

5. EXPERIMENTS

In this section we present our experiments with the two implementations of SSNA, and we compare the two approaches. The experiments were conducted on a Dell Latitude with Intel Core 2 Duo P8400 processor and with 4 GB of RAM. We used MySQL 5.1 and Neo4j 1.1 as the underline database-management systems, both running locally on the computer that was used for the tests.

5.1 Datasets

In our experiments, we used data that is partially real and partially synthetic. For the geographic network, we took real data of the city Haifa. We generated a geographic hierarchy of depth four: City → Neighborhood → Street → House. A type attribute was added to each geographic object (e.g., house, restaurant, hotel, etc), according to a given distribution that simulates real scenarios. Adjacent objects were connected by an edge, e.g., house number 3 in some street was connected to house Number 1 and to house Number 5 in that street. For the social network, we generated users with attributes `occupation` and `hobbies`. The values of these attributes and friendship relationships were added randomly.

The life patterns were also generated synthetically. We used several types of typical patterns: *every morning in every day*, as a daily pattern; *every morning in Monday and Thursday*, as a weekly pattern; and *every night in workdays*, as a weekly pattern. The patterns were generated using the following parameters. (1) Each user has 10 to 30 life patterns. (2) For 97% of the users we generated a 'home' pattern—a pattern that associates the user to some house, for every night of the week. (3) For 80% of the users, a 'workplace' pattern was created—a pattern that indicates being in some location, for several hours on every workday. (3) The locations of patterns were chosen randomly, choosing from all the levels of the geographic hierarchy.

For the tests, we built four socio-spatial networks of different sizes. Their parameters are depicted in Table 1.

Network number	Geographic Objects	Users	Friends per user	Life patterns
I	23493	1931	15	50605
II	47043	3862	30	197425
III	70222	5793	45	441375
IV	92403	13517	60	786502

Table 1: Sizes of the networks.

5.2 Results

The first set of experiments we present examines the operators `Extend` and `Bridge`. We compared the evaluation times, for these two operators, in the two different implementations. For each operation, we examined applying it 1, 2 or 3 times. The expressions are presented in Table 2.

Since both Neo4j and MySQL have a caching mechanism, we tested each query for two cases. In the first case, the cache is empty at the beginning of the test (the systems was shutdown and restarted before the test). In the second experiment, the results are measured after the system has run for a while and the queries were executed several times. In Table 3, the evaluation times are presented. Since Network I is very small, we only present the results for networks II, III, IV. The row for E1 and Network II refers to applying `Extend(..., 1)` over Network II of Table 1. The row for B2 and Network II refers to applying `Bridge(Bridge(...), ...)` (i.e., the second expression of Table 2) over Network II of Table 1. The meaning of the other rows is similar. Each case was tested for several different initial sets of nodes, in

	Bridge	Extend
1	`Bridge(Select(N,name=‘John’),*,0.8)`	`Extend(Select(N,name=‘John’),1)`
2	`Bridge(Bridge(Select(N,name=‘John’),*,0.8),*,0.8)`	`Extend(Select(N,name=‘John’),2)`
3	`Bridge(Bridge(Bridge(Select(N,name=‘John’),*,0.8),*,0.8 ),*,0.8)`	`Extend(Select(N,name=‘John’),3)`

Table 2: The tested expressions: Bridge (B1, B2, B3), and Extend (E1, E2, E3).

different runs.

In the results, we can see the dramatic effect of the cache on the evaluation time. In the evaluation of `Bridge`, Neo4j outperforms MySQL for large networks. The cache increases the efficiency over both platforms. In the evaluation of `Extend`, the two implementations are efficient when caches are being used, however, MySQL is highly inefficient without a cache. This is because `Extend` is translated to a sequence of joins, and join is an expensive relational operation.

	Net-work	Result size	Neo4j		MySQL	
			W/O Cache	With Cache	W/O Cache	With Cache
B1	II	13	766	0	313	0
	III	25	641	0	578	0
	IV	42	484	0	515	0
B2	II	344	23766	32	1234	0
	III	5553	30360	1546	3547	750
	IV	7157	29406	3766	4563	546
B3	II	209	5031	0	125	0
	III	34359	8218	3453	27359	19171
	IV	52235	19782	13672	66328	48016
E1	II	74	0	0	219	0
	III	43	0	0	344	0
	IV	130	0	0	328	0
E2	II	2593	172	78	1625	0
	III	2633	94	109	875	16
	IV	6465	609	859	3422	31
E3	II	3862	1312	515	9047	735
	III	5793	1297	1156	26578	782
	IV	7724	4735	4610	65719	5031

Table 3: The evaluation time (in milliseconds) of Bridge (B1, B2, B3) and Extend (E1, E2, E3).

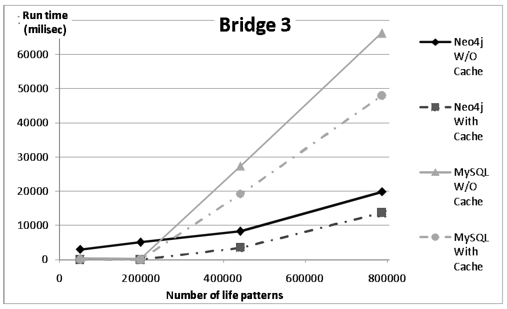

Figure 1: Evaluation time of Bridge 3 as a function of the number of life-pattern edges.

Figure 1 illustrates the effect of the number of life-pattern edges on the `Bridge` operator. Up to a certain limit, the evaluation can be done from the cache of the system (memory) and we do not see any significant increase in evaluation time when the number of life-pattern edges increases. However, when the number of edges exceeds the limit, we see an increase in the evaluation time. The increase is more significant for a relational database than for a graph database because the graph database utilizes better the graph structure of the data.

Figure 2 shows the increase in the evaluation time of `Extend` as a function of the number of users in the social network. Note that the effect on a relational database is more significant than the effect on a graph database.

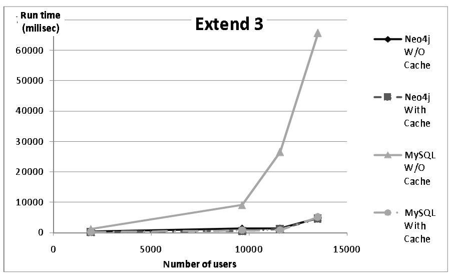

Figure 2: Evaluation time of Extend 3 as a function of the number of users in the network.

In the following experiments, we compare the evaluation of two complex queries. The first query (Query 1) finds locations where it is likely that a paramedics lives there.

```
Paramedics = Select($N_{social}$,
      occupation=‘Paramedics’)
Query_1 = Bridge(Paramedics,
      ‘some_night_of_the_week’,0.85)
```

The second query (Query 2) finds people that are related to John in the following way. They frequently visit in 20%, or more, of the places that John frequently visits. In the query, we use the time pattern *all* as the union of all the time patterns in the SSN.

```
John = Select($N_{social}$, name=‘John’)
Places = Bridge(John, all, 0.5)
Query_2 = MBridge(Places, all, 0.5, 20%)
```

The execution times of these queries are presented in Table 4 and in figures 3 and 4. We can see that caching has a significant effect on both Neo4j and MySQL, especially for Query 1. For Query 2, the Multi Bridge operator requires the computation of aggregate functions, and this is being done effectively by MySQL. Thus, for Query 2, over large networks MySQL is more efficient than Neo4j.

6. RELATED WORK

Recently, the importance of a synergy between social services and GIS tools has been recognized [1]. Huang and

	Network	Result size	Neo4j W/O Cache	Neo4j With Cache	MySQL W/O Cache	MySQL With Cache
Q1	I	11	94	0	281	0
	II	16	235	0	454	0
	III	26	438	0	843	0
	IV	69	922	31	1344	0
Q2	I	1	125	0	109	0
	II	1	453	0	109	0
	III	367	1734	188	938	406
	IV	1	2641	1516	1547	625

Table 4: Evaluation times of Query 1 (Q1) and Query 2 (Q2), in milliseconds.

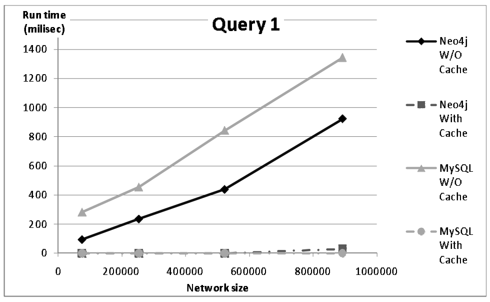

Figure 3: Evaluation time of Query 1 (in milliseconds) as a function of the network size.

Liu [3] introduced the concept of geo-social network services. They showed how social networks could benefit from relating them to spatial applications, and they suggested applications for geo-social network services. Li and Chen [2] analyzed location-based social networks (LBSN). They experimented with Brightkite—a location-based social networking website that is accessible via mobile devices. Users can "check in" to the site and see nearby friends, according to the friendship relations in the social network. They analyzed user behavior in such network and studied the problem of predicting user location based on logged location history. Li and Chen [6] provided a three-layer friendship model for analyzing Brightkite. One layer is based on the location of users, the second is based on the network connections and the third is based on the tags of their profiles. They tested different types of correlations between users, based on the three types of layers, and compared these correlation types.

Zheng et al. [9] presented Geo-life 2.0—a location-based social network service that is based on a GPS log. Li et al. [4] showed how to mine a GPS log and find locations that are point-of-interest. They also studied the problem of finding similarity among users based on points where users stayed for a relatively long time, according to their GPS logs [10]. Zheng et al. [8] showed how to find meaningful life patterns form GPS logs.

7. CONCLUSIONS

The ability to combine social information with spatial information can be useful in different scenarios, such as for research, for social purposes and as part of a statistical or economical analysis. Our main contributions are (1) showing how to combine social information, from a social network, with spatial information, by using life-pattern edges to associate the two networks; (2) presenting a socio-spatial network model and an algebra to effectively query the integrated data; and (3) demonstrating the feasibility of our approach by implementing the model.

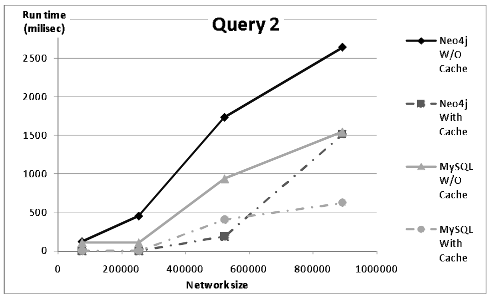

Figure 4: Evaluation time of Query 2 (in milliseconds) as a function of the network size.

Future work includes the development of a declarative language over the proposed algebra, and strengthening the language by including in it aggregation functions and the ability to apply selection based on complex time constraints.

8. REFERENCES

[1] M. F. Goodchild. Social Sciences: Interest in GIS Grows. `http://www.esri.com/news/arcnews/spring04articles/social-sciences.html`, 2004.

[2] Q. Huang and Y. Liu. Analysis of a location-based social network. In *International Conference on Computational Science and Engineering*, pages 263–270, Washington, DC, USA, 2009.

[3] Q. Huang and Y. Liu. On geo-social network services. In *17th International Conference on Geoinformatics*, pages 1–6, 2009.

[4] Q. Li, Y. Zheng, X. Xie, Y. Chen, W. Liu, and W. Ma. Mining user similarity based on location history. In *16th International Conference on Advances in Geographic Information Systems*. ACM, 2008.

[5] MySQL. `http://www.mysql.com/`.

[6] L. Nan and C. Guanling. Multi-layered friendship modeling for location-based mobile social networks. In *6th International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous)*, pages 1–10, 2009.

[7] Neo4j. `http://neo4j.org/`.

[8] Y. Ye, Y. Zheng, Y. Chen, J. Feng, and X. Xie. Mining individual life pattern based on location history. In *10th International Conference on Mobile Data Management*, pages 1–10, Taipei, Taiwan, 2009. IEEE Computer Society.

[9] Y. Zheng, Y. Chen, X. Xie, and W. Ma. GeoLife2.0: A location-based social networking service. In *10th International Conference on Mobile Data Management: Systems, Services and Middleware*, pages 357–358, 2009.

[10] Y. Zheng, L. Zhang, X. Xie, and W. Ma. Mining interesting locations and travel sequences from gps trajectories. In *18th International Conference on World Wide Web*, pages 791–800. ACM, 2009.