

Optimizing Online Spatial Data Analysis with Sequential Query Patterns

Chunqiu Zeng, Hongtai Li, Huibo Wang, Yudong Guang,
Chang Liu, Tao Li, Mingjin Zhang, Shu-Ching Chen, Naphtali Rishe
School of Computing and Information Sciences,
Florida International University

{czeng001, hli019, hwang033, yguan004, cliu019, taoli, zhangm, chens, rishen}@cs.fiu.edu

Abstract—The exponential growth of the usage of geographic information services leads to an extensive popularity on spatial data analysis in both industrial and academic communities. However, it is quite challenging for users to efficiently analyze and quickly understand the spatial data due to the inherently complex and dynamic nature of GIS applications.

To address the challenges, this paper presents an approach to optimize the online spatial data analysis by mining the sequential query patterns from the user query logs of GIS applications. The sequential query patterns are used to automatically generate the query template, from which the users are able to quickly compose new queries. The sequential query patterns contribute to the workflow construction for complex spatial data analysis tasks as well. Our proposed approach takes advantage of the generated workflow to parallelize the independent spatial analysis tasks. As a result, the throughput of our system has been increased greatly and more efficient geographic information services are made available to the users. We present a case study to demonstrate the efficiency and effectiveness of the proposed approach by integrating two software systems at Florida International University (FIU): TerraFly (an GIS application) and FIU-Miner (a distributed data mining framework).

I. INTRODUCTION

With the rapid advancement in technology of geographic information system, online spatial data analysis becomes increasingly essential in various application domains such as water management, crime mapping, disease analysis, and real estate. As a consequence, many geographic applications from different domains emerge recently in the form of web applications or mobile applications. Miscellaneous requirements from diverse application domains strongly dictate efficient support for spatial data analysis.

However, the inherently complex and dynamic nature of GIS applications gives rise to great challenges for users to efficiently analyze and quickly understand the spatial data. Spatial data analysis conducted on a typical geographic application tends to involve a series of complicated interactions and may be fussy with a lot of low-level details. In addition, users from different domains want GIS systems to dynamically create map applications on their own spatial data sets. Moreover, massive spatial data analysis conducted on GIS applications is very resource-consuming. These big challenges require many GIS applications to be designed with innovative approaches to gain competitive advantages.

Recently, TerraFly GeoCloud is designed and developed to support spatial data analysis and visualization [7]. Point and

polygon spatial data can be accurately visualized and manipulated in TerraFly GeoCloud. It allows users to visualize and share spatial data related to different domains such as real property, crime, and water resources. Online analysis of spatial data is supported by the spatial data analysis engine of TerraFly GeoCloud as well. In order to efficiently support complex spatial data analysis and flexible visualization of the analysis results, MapQL, a SQL-like language, is implemented to represent the analysis queries in TerraFly GeoCloud. A MapQL statement is capable of defining an analysis task and customizing the visualization of analysis results. According to the queries, the spatial data analysis engine completes the analysis task and renders the customized visualization of analysis results. For instance, given the real property data, a user may want to explore the house prices near Florida International University. The corresponding MapQL statement for such an exploration is shown in Figure 1.

```
SELECT
    '/var/www/cgi-bin/house.png' AS T_ICON_PATH,
    r.price AS T_LABEL,
    '15' AS T_LABEL_SIZE,
    r.geo AS GEO
FROM
    realtor_20121116 r
WHERE
    ST_Distance(r.geo, GeoFromText('POINT(-80.27, 25.757228)')) < 0.3;
```

Fig. 1: A MapQL query on real property data is given, where POINT(-80.27,25.757228) is the location of Florida International University.

A MapQL statement extends the semantics of traditional SQL statements by introducing new reserved key words. As shown in Figure 1, *T_ICON_PATH*, *T_LABEL*, *T_LABEL_SIZE* and *GEO* are four additional reserved words in a MapQL statement. These four reserved key words are used in the “*expression AS < reserved word >*” clause, which provides the expression with additional semantics. In particular, *GEO* describes the spatial search geometry; *T_ICON_PATH* customizes the icon resource for the spatial search geometry; *T_LABEL* provides the icon label to be shown on the map; and *T_LABEL_SIZE* gives the size of

label in pixels. The corresponding spatial query results for the MapQL statement in Figure 1 is presented in Figure 2.

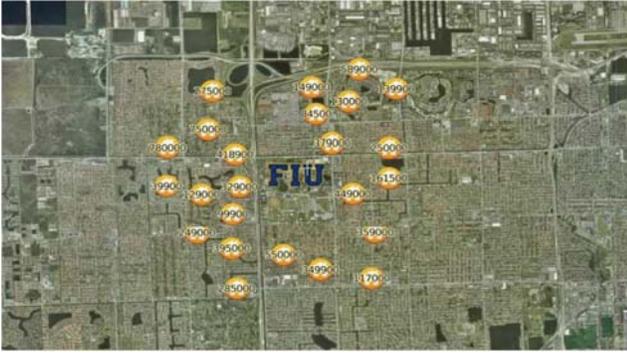


Fig. 2: The MapQL query result on real property data is displayed on the map.

Comparing with using GIS application programming interface (API), MapQL provides a better interface to facilitate the use of TerraFly map for both developers and end users without any functionality limitation. Similar to GIS API, MapQL enables users to flexibly create their own maps. However, our further study of TerraFly GeoCloud reveals three interesting and crucial issues which present similar challenges in other online spatial analysis systems.

The first issue is the difficulty in authoring MapQL queries. Though most of developers who are familiar with SQL can pick up MapQL quickly, the learning curve for end users who have no idea about SQL before is very steep. Authoring MapQL queries remains a challenges for the vast majority of users. As a result, it is difficult for those end users to utilize MapQL to complete a spatial analysis task from scratch.

The second issue is the complexity of a spatial analysis task. A typical spatial analysis task tends to involve a few sub-tasks. Moreover, those sub-tasks are not completely independent from each other, where the outputs of some sub-tasks are used as the inputs for other sub-tasks. According to the dependencies, a spatial data analysis task can be naturally presented as a workflow. The complexity of building such a workflow turns out to be a great obstacle for the users during the online spatial data analysis.

The third issue is the inefficiency of sequentially executing the workflow of a spatial analysis task. Even though the sub-tasks in a workflow are not linearly dependent on each other, the sub-tasks can only be sequentially executed by end users one by one. As a consequence, it fails to take advantage of the distributed environment to optimize the execution of independent sub-tasks in parallel.

The above three issues pose big challenges for users to freely and flexibly explore spatial data using online spatial analysis system. In this paper, we employ sequential pattern mining algorithms to discover the sequential query patterns from the MapQL query logs of TerraFly GeoCloud. With the help of discovered sequential query patterns, the workflows of spatial data analysis tasks are first constructed. FIU-Miner [13] is then employed to optimize the execution of the spatial data

analysis tasks by maximizing the parallelization of sub-tasks in the corresponding workflow.

The rest of the paper is organized as follows. Section II presents the related work. Section III gives the overview of our system. Section IV introduces the details in mining sequential query patterns among MapQL query logs, the generation of query templates for MapQL statements, and the workflow construction for the spatial data analysis tasks. Section V presents our empirical study to show the efficiency and the effectiveness of our system. Finally Section VI concludes and discusses future works.

II. RELATED WORK

Sequential pattern mining has been studied for decades. Several famous algorithms have been proposed to discover the sequential pattern in sequence data including apriori-based GSP [11], SPIRIT [4], Spade [12], FreeSpan [5], and PrefixSpan [8]. This paper adapts the PrefixSpan algorithm to find the frequent sequential pattern.

Mining patterns in SQL logs has received a lot of attention in recent years, especially with the advent of Big data era. The research efforts in [3], [1] employ the discovered patterns from the SQL logs to facilitate the SQL auto-completion with recommending snippets of query. However, almost all the reported studies only focus on the patterns within a single statement. In this paper, we focus on the sequential query pattern which consists of a sequence of statements.

TerraFly [10], [9] is a platform which supports query and visualization of geo-spatial data. This platform provides users with customized aerial photography, satellite imagery and various overlays, such as street names, roads, restaurants, services and demographic data. TerraFly API allows application developer to create various GIS applications, such as geospatial querying interface, map display with user-specific granularity, real-time data suppliers, demographic analysis, annotation, and route dissemination via autopilots. TerraFly GeoCloud [7] is a system built on the TerraFly system. One of important features of TerraFly GeoCloud is that it provides MapQL to support more flexible and complicated spatial data analysis. However, it requires end users to compose the MapQL statements.

FIU-Miner is a data mining platform which supports data analyst with a user-friendly interface to perform rapid data mining task configuration. Users can assemble the existing algorithms into a workflow without writing a single line of code. The platform also supports data mining in distributed and heterogeneous environments [13]. In this paper, we take advantage of FIU-Miner to optimize the execution of spatial data analysis.

III. THE SYSTEM OVERVIEW

To address the highlighted issues of TerraFly GeoCloud in Section I, the online spatial analysis system is optimized by integrating FIU-Miner framework, which is capable of assembling sub-tasks into a workflow in accordance with the dependencies of sub-tasks and scheduling each sub-task for

execution in distributed environment. The overview of the integrated system is given in Figure 3.

The system consists of four tiers: User Interface, Geo-Spatial Web Service, Computing Service and Storage.

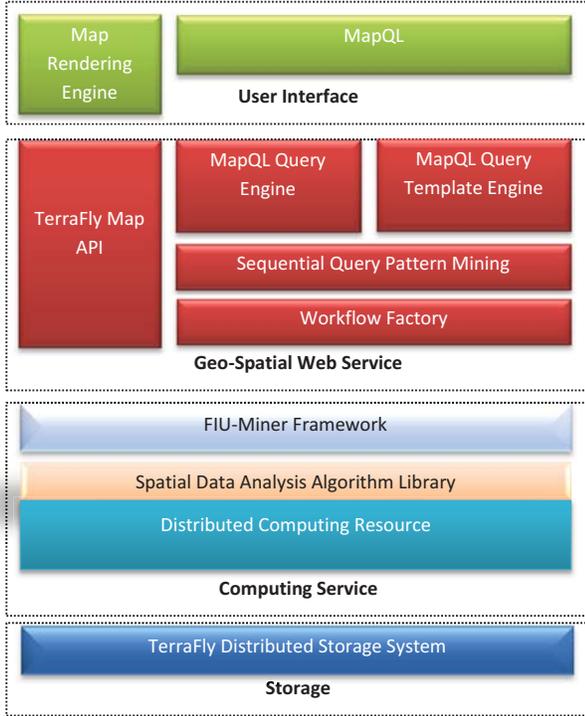


Fig. 3: The system overview.

In the layer of User Interface, Map Rendering Engine is responsible for rendering the geo-objects on the map nicely based on the visualization customized by users. The component of MapQL accepts MapQL statements that describe the spatial analysis task and the required elements for map rendering.

The second layer is Geo-Spatial Web Service. In this layer, TerraFly Map API provides the interface to access the spatial data for other components in the same layer and Map Rendering Engine in User Interface layer. MapQL Query Engine is responsible for analyzing the MapQL statements and guarantees their syntactic and semantic correctness. Sequential Query Pattern Mining is utilized to discover the sequential query pattern from the MapQL query log data. The discovered sequential query pattern can be used to generate the query templates by MapQL Query Template Engine. Users are able to rewrite the MapQL query template to construct new MapQL statements in User Interface layer. A sequential query pattern contains a sequence of MapQL queries and is used to form a workflow by Workflow Factory. Each query in a sequential pattern corresponds to a sub-task in the corresponding workflow.

The third layer is Computing Service. FIU-Miner Framework takes a workflow from the second layer as an input. FIU-Miner takes the load balance of distributed environment into

account to schedule the sub-tasks of a workflow for execution. The spatial data analysis library is deployed in the distributed environment. The library can be extended by developers. The computing resource is used to support the spatial data analysis tasks.

The last layer is mainly responsible for storing and managing the spatial data. All the spatial data in TerraFly is stored in the distributed file system, where replica of data guarantees the safety and reliability of system.

In the subsequent sections, we introduce the detail of the proposed system.

IV. SEQUENTIAL QUERY PATTERN

In our system, users mainly use MapQL statements to accomplish their online spatial data analysis tasks. Although MapQL is powerful and flexible to satisfy the analysis requirement of the users, it requires end users to compose the statements, typically from scratch. Based on the user query logs, the sequential MapQL query pattern is proposed to partially address the problem.

A. Sequential MapQL Query Pattern

Let D be a collection of sequences of queries, denoted as $D = \{S_1, S_2, \dots, S_n\}$, where S_i is a sequence of queries occurring within a session, ordered according to their time stamps. Therefore, $S_i = \langle q_1, q_2, \dots, q_i, \dots, q_m \rangle$ is a sequence including m queries in temporal order. If q_i is a compound query composed of two sub-queries q_{i0} and q_{i1} , then $S_i = \langle q_1, q_2, \dots, (q_{i0}, q_{i1}), \dots, q_m \rangle$. Sub-queries in a parenthesis are from a compound query occurring at the same time stamp.

A k -subsequence of S_i is a sequence of queries with length k denoted as $T = \langle t_1, t_2, \dots, t_k \rangle$, where each $t \in T$ corresponds to only one query $q \in S_i$, and all the queries in T are kept in temporal order. $T \sqsubseteq S_i$ is used to indicate that T is a subsequence of S_i .

Given the query sequence data collection D , a sequential query pattern is a query sequence whose occurrence frequency in the query log D is no less than a user-specified threshold $min_support$. Formally, the support of sequence T is defined as

$$support(T) = |\{S_i | S_i \in D \wedge T \sqsubseteq S_i\}|.$$

A sequence T is a sequential query pattern, only if $support(T) \geq min_support$.

The process of discovering all the sequential query patterns from the MapQL logs generally consists of two stages. The first stage is to generalize the representation of MapQL statements by parsing the MapQL text into syntax units. Based on the syntax representation of MapQL statements, the second stage is to mine the sequential query patterns from the sequences of MapQL statements.

B. Representation Of MapQL

As shown in Figure 3, MapQL Query Engine collects the MapQL statements and records them in the log files. A snippet of MapQL logs are given in Table I. Each MapQL statement is associated with a user session ID and a time stamp. All

the statements are organized in temporal order. Those MapQL statements sharing the same session ID are those issued by a user within a session. Our goal is to discover interesting patterns from the query logs. For example, according to the log data in the TABLE I, an interesting pattern is that users who viewed a particular street are more likely to look for the hotels along that street.

TABLE I: A snippet of MapQL logs.

Session ID	Timestamp	MapQL statement
1	20140301 13:26:33	SELECT geo FROM street WHERE name LIKE 'sw 8th';
1	20140301 13:28:26	SELECT h.name FROM street s LEFT JOIN hotel h ON $ST_Distance(s.geo, h.geo) < 0.05$ WHERE s.name = 'sw8 th' AND h.star ≥ 4 ;
2	20140315 14:21:03	select geo from street where name like 'turnpike';
2	20140315 14:25:21	SELECT h.name FROM street s LEFT JOIN hotel h ON $ST_Distance(s.geo, h.geo) < 0.05$ WHERE s.name = 'turnpike' AND h.star ≥ 4 ;
3	20140316 10:23:08	SELECT zip FROM us_zip;
4	20140319 11:19:21	SELECT count(*) FROM hotel;
...

In order to discover patterns from the query logs, intuitively, existing sequential pattern mining algorithms can be directly applied to the raw logs of MapQL statements, where different text of MapQL statements are treated as different items. However, representing an query item by the text of the MapQL statement is often too specific. As a consequence, it is difficult to discover the meaningful sequential patterns with such representations. For instance, the first and third records in TABLE I are identified as different query items during sequential query pattern mining, although both MapQL statements share the same semantics (i.e., locating a street given its partial name).

To address the aforementioned problem, the representation of a query in our system is generalized by parsing a MapQL statement into a collection of syntax units. The syntax units are organized as a syntax tree. For instance, the syntax tree for the first record of TABLE I is presented in Figure 4. There are two types of labels in the node of syntax tree. One is the type of a syntax unit, such as “Select Clause”. The other label in the parenthesis is the content of a syntax unit, for example, “sw 8th”. Provided with the syntax tree, the MapQL query can be generalized by representing any nodes with their types instead of their actual contents. For instance, assuming the node with ‘Value’ type in the syntax tree is represented as “#Value#” rather than using its text content, the original MapQL statements in both the first and third row of TABLE I are rewritten as “SELECT geo FROM street WHERE name LIKE #Value#;”. Therefore, the two MapQL statements with the same semantics share the same query item. In addition, to

simplify the extraction of patterns, each query item is identified with a unique integer number.

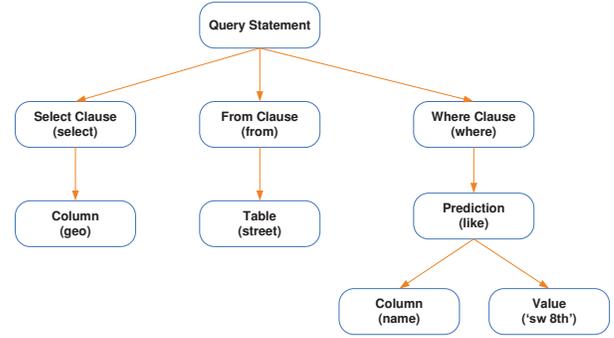


Fig. 4: The syntax tree for a MapQL statement “SELECT geo FROM street WHERE name LIKE ‘sw 8th’;”.

C. Mining Sequential Query Pattern

Based on the properly generalized representation of a MapQL query, the PrefixSpan algorithm [8] is applied to efficiently discover all the sequential query patterns from the MapQL query log data.

The main idea of the PrefixSpan algorithm is to recursively partition the whole dataset into some sub-datasets, where the query sequences in a sub-dataset share the same prefix subsequence. The number of query sequences in each sub-dataset indicates the *support* of its corresponding prefix subsequence. If a prefix subsequence T whose *support* is no less than the user specified threshold $min_support$, T is a sequential query pattern. Given two sequences T and R , $T \sqsubseteq R$ if T is a subsequence of R . An important property (i.e., downward closure property) is that R cannot be a sequential query pattern if T is not a sequential query pattern. According to the property, the recursive partition to search for super-pattern is not terminated until the size of current sub-dataset is smaller than the $min_support$.

The PrefixSpan algorithm is illustrated in Figure 5. The top table presents the original collection of query sequences which contains two sequences of queries $S_1 = \langle (q_0, q_1) \rangle$ and $S_2 = \langle (q_0), (q_2) \rangle$. Sequence S_1 has only one compound query composed of q_0 and q_1 . The other sequence S_2 has two queries named q_0 and q_2 . Let $min_support = 2$. The procedure of mining sequential query patterns is described as follows.

- 1) Find the frequent subsequences with a single query: $\langle q_0 \rangle$, $\langle q_1 \rangle$, $\langle q_2 \rangle$.
- 2) Take the subsequences found in the step 1 as the prefixes. Based on the prefixes, the original dataset is partitioned into three sub-datasets, where each of them is specified by a prefix subsequence. The support of the prefix subsequence is the number of postfix sequences in its corresponding sub-dataset. The prefix patterns are extracted if their supports are larger than $min_support$. Only the prefix subsequence in $D1$ is a sequential query pattern.

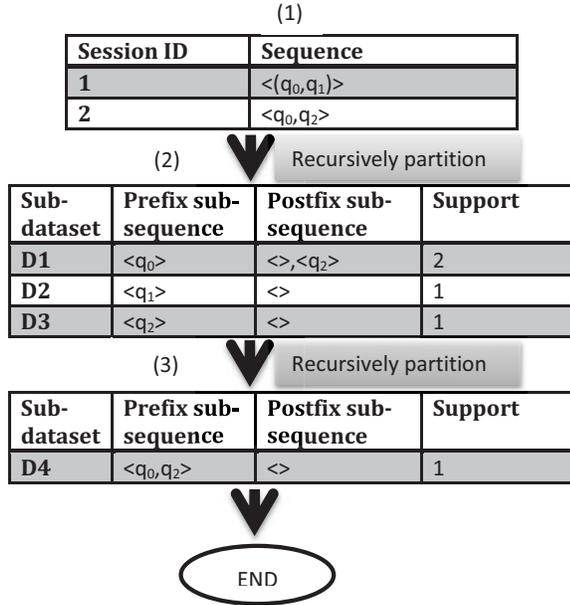


Fig. 5: An example illustrating the PrefixSpan algorithm.

- 3) Recursively partition $D1$. As a result, only one sub-dataset is generated and its support is 1.
- 4) Terminate the partition since no new prefix patterns can be further derived.

In the end, PrefixSpan discovers one sequential query pattern $\langle q_0 \rangle$.

D. Query Template

Query template is generated by MapQL Query Template Engine in the system. This function alleviates the burden of users since MapQL queries can be composed by rewriting query templates. Based on the discovered sequential query patterns, a query template is generated by Algorithm 1. This algorithm scans the syntax trees in the sequential query pattern and replaces the specific table, column and constant value with template parameters. The algorithm guarantees that the same table, column or constant value appearing at multiple places, even multiple queries of a sequence acquires the same template parameter. Users can easily convert the template to executable queries by assigning the template parameters with specific values.

Given a sequential query pattern that contains the two queries with session ID 1 in TABLE I, we can apply Algorithm 1 to generate the template for the sequential query pattern. The generated template is shown in Figure 6. This template owns three parameters (i.e., #arg1#, #arg2#, #arg3#). Provided with values of these parameters, the executable sequence of queries can be easily derived from the template.

E. Spatial Data Analysis Workflow

All the MapQL queries in a sequential pattern are organized in a workflow, where the template parameters indicate the data transmission between queries. A sequence of queries

Algorithm 1 templateGen

- 1: **procedure** *templateGen*(S)
 - ▷ **Input:** S is a sequential pattern which contains a sequence of MapQL queries.
 - ▷ **Output:** query template for sequential query pattern.
- 2: initialize an empty inverted index TAB for Table
- 3: initialize an empty inverted index COL for Column
- 4: initialize an empty inverted index VAL for Value
- 5: **for** each query q_i in S **do**
- 6: **while** each node e in the syntax tree of q_i **do**
- 7: **if** $e.label$ is not generalized **then**
- 8: CONTINUE
- 9: **end if**
- 10: **if** $e.type$ is TABLE **then**
- 11: add element to TAB for the table name
- 12: **end if**
- 13: **if** $e.type$ is COLUMN **then**
- 14: add element to COL for the column name
- 15: **end if**
- 16: **if** $e.type$ is VALUE **then**
- 17: add element to VAL for the constant value
- 18: **end if**
- 19: **end while**
- 20: **end for**
- 21: With TAB, COL, VAL , rewrite S by replacing a specific table, column and constant value with corresponding template name.
- 22: **return** the rewritten sequence as template.
- 23: **end procedure**

```

Template(#arg1#, #arg2#, #arg3#):

SELECT    geo
FROM      street
WHERE

        name LIKE #arg1#;

SELECT    h.name
FROM      street s
LEFT JOIN hotel h
ON        ST_Distance(s.geo,h.geo) < #arg2#
WHERE    s.name LIKE #arg1#
AND      h.star >= #arg3#;

```

Fig. 6: Example of a generated template.

constitutes a spatial data analysis task and atypical spatial data analysis task often involves a few sub-tasks. The dependencies among those sub-tasks make spatial data analysis very complicated. The complexity of spatial data analysis dictates the support of workflow. In our system, Workflow Factory is designed and implemented in support of executing a complex spatial data analysis task in a workflow. A workflow is represented as a directed and connected graph consisting of nodes (denoting the sub-tasks) and edges (describing the dependencies among the sub-tasks). Data transmission between dependent sub-tasks

are supported in our system.

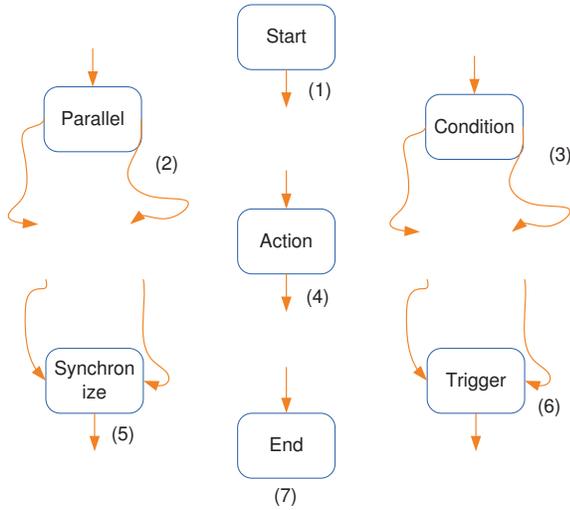


Fig. 7: Node types in a workflow.

In order to facilitate the spatial data analysis, we design seven types of nodes as shown in Figure 7.

- 1) **Start Node** This type of node indicates the start of the workflow. There is only one such node in a valid workflow. This start node must link to one other node.
- 2) **Parallel Node** This type of node has one input link and more than one output links. After the work is completed in the parent node, the parallel node triggers the sub-tasks in its children nodes. All the sub-tasks of its children are executed in parallel.
- 3) **Condition Node** One input link and more than one output links are associated with this type of node. When the control flow reaches a condition node, it will check the input data and then move along one of its output links.
- 4) **Action Node** One input link and one output link are associated with this type of node. It often accommodates the sub-tasks for spatial data analysis. The data from the input link is fed into the sub-task and the result data of this sub-task is forwarded along its output link.
- 5) **Synchronize Node** This type of node has more than one input links and one output link. This node does not direct the control flow to its output link until all the sub-tasks in its parent nodes is completed.
- 6) **Trigger Node** More than one input links and one output link are associated with this type of node. The node starts the sub-tasks in its output link once one of sub-tasks in its parent nodes is finished.
- 7) **End Node** Any valid workflow should have one and only one end node. It indicates the end of the workflow.

Based on the generated template in Figure 6, two simple workflows can be constructed in Figure 8. These two workflows accomplish the same spatial data analysis task described in Figure 6. In the subfigure (1) of Figure 8, the two sub-tasks (i.e., SearchStreet and SearchHotel) are executed sequentially.

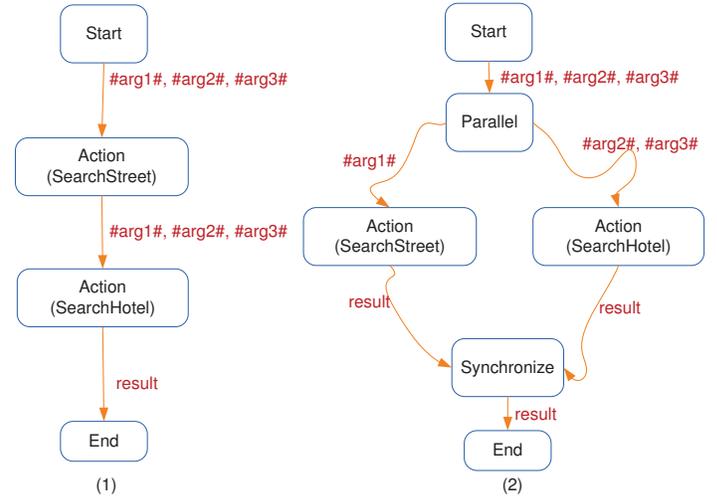


Fig. 8: Workflow examples.

However, SearchStreet needs the template parameter #arg1# as its input, while SearchHotel needs all three parameters. Provided with the three parameters, both sub-tasks can be executed independently. Thus, in the subfigure(2) of Figure 8, a parallel workflow is introduced to complete the spatial data analysis task. Since our data analysis tasks are scheduled by FIU-Miner, which takes full advantage of the distributed environment, the parallel workflow is more preferable to our system in terms of efficiency.

V. EMPIRICAL STUDY

In this section, the empirical study is conducted to demonstrate the efficiency and effectiveness of our system.

A. Setup

Besides providing the powerful API to support GIS applications, the TerraFly platform has a rich collection of GIS datasets. TerraFly owns the US and Canada roads data, the US Census demographic and socioeconomic data, the property lines and ownership data of 110 million parcels, 15 million records of businesses with company stats and management roles and contacts, 2 million physicians with expertise detail, various public place datasets, Wikipedia, extensive global environmental data, etc. Users can explore these datasets by issuing MapQL queries in our system. A case study on house property exploration is conducted to show how our system works.

B. House Property Exploration

The proposed system provides an optimized solution to spatial data analysis problem by explicitly constructing a workflow. It supports many different applications by analyzing the corresponding datasets. One typical application scenario is to locate the house property with a good appreciation potential for investment. Intuitively, it is believed that a property is deserved for investment if the price of the property is lower than the ones of surrounding properties. Our system is capable

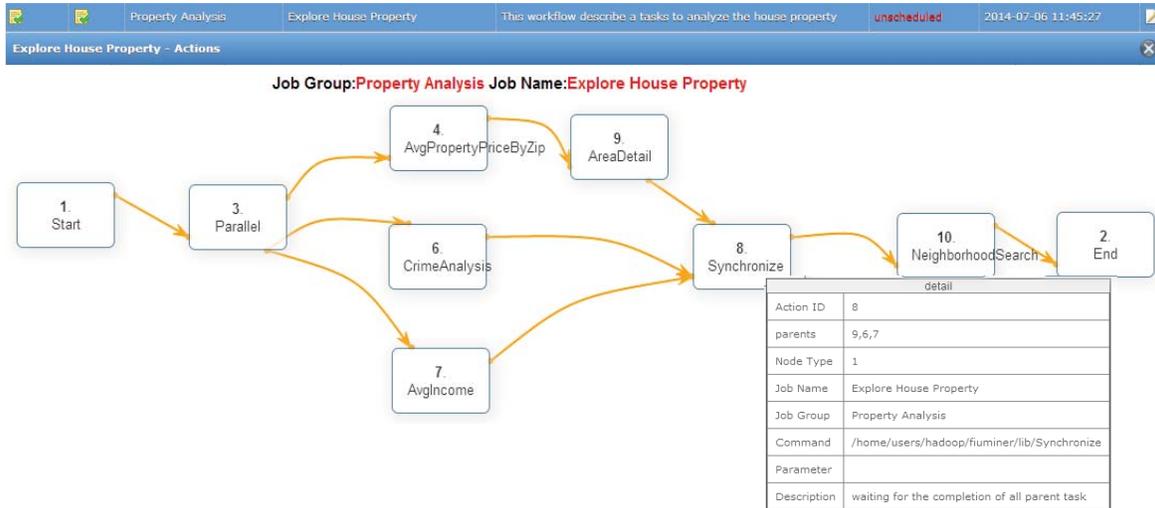


Fig. 9: The workflow of searching for house properties with a good appreciation potential. All the sub-tasks in the workflow are scheduled by FIU-Miner and are executed in the distributed environment.

of helping users (e.g., investors) to easily and conveniently identify such properties. According to the historical query logs collected in our system, the sequential query patterns are extracted. Based on the discovered sequential query patterns, the query templates are then generated automatically. The templates related to the house property case study are assembled to build a workflow for house property data analysis. The workflow is presented in Figure 9.

In the workflow, there are nine sub-tasks, denoted as rectangles, to constitute the complete house property analysis task. A user can view the detailed information of each sub-task from a pop-up layer as long as the mouse hovers on the corresponding node. The workflow begins with a start node, which is used to prepare the required setting and parameters. The start node links to the parallel node with three out links. The parallel node indicates that the three sub-tasks along its out links are able to be executed simultaneously.



Fig. 10: Average property prices by zip code in Miami.

The AvgPropertyPriceByZip node in the workflow calculates the average property price. The overview of the analysis results is presented in Figure 10. Note that the property prices of red regions are higher than those of blue regions. From the overview, users often interested in the regions marked with a

green circle since the average property price of the region is lower than the ones of its surroundings.



Fig. 11: Detailed properties in Miami.

In the next step, users check more detailed information on the region in the green circle by conducting the data analysis in the AreaDetail node. The spatial auto-correlation analysis on the average property prices by zip code data in Miami is conducted in this node and the analysis results are shown in Figure 11. Each point in the scatter plot corresponds to one zip code. Moran's I measure is applied during the auto-correlation analysis [2], [6]. The points in the first and third quadrants show positive associations with its surroundings, while the points in the second and fourth quadrants indicate negative associations. Herein, users are generally interested in the points of second quadrants, having lower property price than the ones of its surrounding regions. The interesting points are marked with yellow circles. The analysis leads to the result that most of the cheap properties with good appreciation potential are along the Gratigny Pkwy.

In order to make sure that the areas with cheap properties have good appreciation potential, spatial data analysis to investigate the crime rate and average income of these areas are conducted. The two data analysis sub-tasks are described in CrimeAnalysis and AvgIncome nodes, respectively. These two

sub-tasks are executed in parallel with the properties analysis. The Synchronize node waits for the completion of all three sub-tasks along the in links. Parallel execution accelerates the whole spatial data analysis and reduces the time cost.

```

SELECT
CASE
WHEN h.pvalue >= 400000
THEN '/var/www/cgi-bin/redhouse.png'
WHEN h.pvalue BETWEEN 200000 AND 400000
THEN '/var/www/cgi-bin/bluehouse.png'
WHEN h.pvalue BETWEEN 100000 AND 200000
THEN '/var/www/cgi-bin/bluehouse.png'
ELSE '/var/www/cgi-bin/darkhouse.png'
END AS T_ICON_PATH,
h.geo AS GEO
FROM
osm_fl o
LEFT JOIN
south_florida_house_price h
ON
ST_Distance(o.geo, h.geo) < 0.05
WHERE
o.name = #arg1# AND
h.std_pvalue < 0 AND
h.std_sl_pvalue > 0;

```

Fig. 12: A template for searching the neighborhood, given the partial name of street.

Without discovering any abnormalities in the crime rate and average income, users proceed to acquire more detailed property information along the Gratigny Pkwy by executing the sub-task in the NeighborhoodSearch node. The MapQL query listed in Figure 12 is executed in the NeighborhoodSearch node by passing the ‘Gratigny Pkwy’ as the input parameter. The MapQL statement employs different colors to mark the regions with various property prices. The final analysis results are presented in Figure 13. The regions painted in dark have the cheapest property prices and good appreciation potential.



Fig. 13: The final analysis results.

With the completion of the sub-task in the NeighborhoodSearch node, the control flow reaches the end node of the workflow. Comparing to the analysis procedure without workflow, where sub-tasks can only be executed sequentially, our system takes full advantage of FIU-Miner to schedule multiple tasks simultaneously in the distributed environments. It greatly reduces the time consumed by a complex spatial data analysis task and increases the throughput of our system.

VI. CONCLUSION

This paper proposes an approach to optimize spatial data analysis by integrating the FIU-Miner framework and the TerraFly system. Our approach makes use of sequential query patterns, which are discovered from the query logs, to facilitate the data analysis with query templates and optimized workflows. A case study is presented to demonstrate the effectiveness and efficiency of our system.

There are several future research directions to improve our current system such as developing more efficient and effective algorithms for discovering the query patterns and designing better techniques for generating query templates and constructing workflows.

ACKNOWLEDGMENT

The work is partially supported by the National Science Foundation under grants CNS-0821345, HRD-0833093, IIP-0829576, CNS-0959985, CNS-1126619, IIP-1338922, IIP-1237818, IIP-1330943, and IIS-1213026, the U.S. Department of Homeland Security under Award Number 2010-ST-062000039, the U.S. Department of Homeland Security's VACCINE Center under Award Number 2009-ST-061-CI0001, and Army Research Office under grant number W911NF-1010366 and W911NF-12-1-0431.

REFERENCES

- [1] Javad Akbarnejad, Gloria Chatzopoulou, Magdalini Eirinaki, Suju Koshy, Sarika Mittal, Duc On, Neoklis Polyzotis, and Jothi S Vindhiya Varman. Sql querie recommendations. *VLDB*, 3(1-2):1597–1600, 2010.
- [2] Luc Anselin. Local indicators of spatial association. *Geographical analysis*, 27(2):93–115, 1995.
- [3] Gloria Chatzopoulou, Magdalini Eirinaki, and Neoklis Polyzotis. Query recommendations for interactive database exploration. In *Scientific and Statistical Database Management*, pages 3–18, 2009.
- [4] Minos N Garofalakis, Rajeev Rastogi, and Kyuseok Shim. Spirit: Sequential pattern mining with regular expression constraints. In *VLDB*, volume 99, pages 7–10, 1999.
- [5] Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. Freespan: frequent pattern-projected sequential pattern mining. In *ACM SIGKDD*, pages 355–359, 2000.
- [6] Hongfei Li, Catherine A Calder, and Noel Cressie. Beyond moran's i: testing for spatial dependence based on the spatial autoregressive model. *Geographical Analysis*, 39(4):357–375, 2007.
- [7] Yun Lu, Mingjin Zhang, Tao Li, Yudong Guang, and Naphtali Rishe. Online spatial data analysis and visualization system. In *SIGKDD Workshop on Interactive Data Exploration and Analytics*, pages 71–78, 2013.
- [8] Jian Pei, Helen Pinto, Qiming Chen, Jiawei Han, Behzad Mortazavi-Asl, Umeshwar Dayal, and Mei-Chun Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *ICDE*, pages 215–224, 2001.
- [9] N Rishe, M Gutierrez, A Selivonenko, and S Graham. TerraFly: A tool for visualizing and dispensing geospatial data. *Imaging Notes*, 20(2):22–23, 2005.
- [10] Naphtali Rishe, Shu-Ching Chen, Nagarajan Prabakar, Mark Allen Weiss, Wei Sun, Andriy Selivonenko, and D Davis-Chu. TerraFly: A high-performance web-based digital library system for spatial data access. In *ICDE*, pages 17–19, 2001.
- [11] Ramakrishnan Srikant and Rakesh Agrawal. *Mining sequential patterns: Generalizations and performance improvements*. Springer, 1996.
- [12] Mohammed J Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine learning*, 42(1-2):31–60, 2001.
- [13] Chunqiu Zeng, Yexi Jiang, Li Zheng, Jingxuan Li, Lei Li, Hongtai Li, Chao Shen, Wubai Zhou, Tao Li, Bing Duan. Fiu-miner: a fast, integrated, and user-friendly system for data mining in distributed environment. In *ACM SIGKDD*, pages 1506–1509, 2013.