G-Path: Flexible Path Pattern Query on Large Graphs

Yiyuan Bai[1,2] Chaokun Wang[2] Yuanchi Ning[2] Hanzhao Wu[2] Hao Wang[1,2]
[1]Department of Computer Science and Technology, Tsinghua University
[2]School of Software, Tsinghua University, Beijing 100084, China
eldereal@gmail.com, chaokun@mail.tsinghua.edu.cn,
{ningyc09, wuhz09, wanghao07}@mails.tsinghua.edu.cn

ABSTRACT
With the socialization trend of web sites and applications, the techniques of effective management of graph-structured data have become one of the most important modern web technologies. In this paper, we present a system of path query on large graphs, known as G-Path. Based on Hadoop distributed framework and bulk synchronized parallel model, the system can process generic queries without preprocessing or building indices. To demonstrate the system, we developed a web-based application which allows searching entities and relationships on a large social network, e.g., DBLP publication network or Twitter dataset. With the flexibility of G-Path, the application is able to handle different kinds of queries. For example, a user may want to search for a publication graph of an author while another user may want to search for all publications of the author's co-authors. All these queries can be done by an interactive user interface and the results will be shown in a visual graph.

Categories and Subject Descriptors
H.3.5 [**Information Storage and Retrieval**]: Online Information Services - Web-based services; H.2.5 [**Database Management**]: Languages - Query languages

Keywords
G-Path, regular path pattern, path pattern query, graph query language, social network.

1. INTRODUCTION
With the socialization trend of web sites and applications, the techniques of effective graph-structured data management have become one of the most important modern web technologies. Techniques of efficient and effective query processing on graph datasets, especially on graphs with a huge amount of vertices and edges (known as big graphs), keep becoming dramatically important to many product-level solutions to crucial problems (e.g. social network analysis), and gain increasing attention from researchers [1], [2], [3].

Many emerging algorithms directly or indirectly depend on the effective computation of paths of a specific kind between two nodes, e.g. retrieving all paths of length up to L in the GraphGrep algorithm for sub-graph query processing [4], counting label paths in a given graph for classification of chemical compounds [5], and finding out a path in social network analysis, whose edge colors match the pattern specified by a regular expression [6]. This kind of problems can be called path pattern query, a.k.a. path pattern matching, and is one of the most basic operations of graph data management and mining.

To be generalized to fit into different applications, the definition of path patterns should be flexible to define patterns of different lengths and constraints. A path pattern may involve constraints on nodes as well as edges, along with some repetitive patterns of its parts. Regular expression is suitable for the definition of the pattern because it is simple and powerful. It can express various repetitive and constrained patterns in a single-line plain text. The path pattern defined in regular expressions is called **regular path patterns** in this paper.

There are some query languages supporting path queries, but those only support limited regular expressions (e.g. GraphQL [7], SoQL [8]), or only support semantic web data and cannot directly extend to support universal graph (e.g. GLEEN [9], SPARQL [10]). Several graph data management systems (GDMS) emerged these years, such as Neo4j[1], Apache Giraph[2] or Trinity. However, they have the following problems. Neo4j is a GDMS with strong consistency, but it performs badly on a large distributed environment. Giraph is more likely to be a platform rather than a GDMS because it does not have a high level query language. Trinity is a private system for shortest path calculation, and does not support generic queries.

In this paper, we bring forward the design and implementation of G-Path, a regular path query language on graphs, which supports mostly all useful regular expression operators (group operator, parallel operator, Kleene operators, etc.) in a similar syntax. A G-Path query can be compiled into a finite automaton. We make a distributed algorithm to process G-Path queries based on bulk synchronization parallel (BSP) model and propose several optimization methods to improve the performance. Because bulk synchronization parallel model [11] is a de facto standard in graph data management, G-Path can be easily implemented on various graph data management and processing platforms (e.g. Google Pregel [12], Apache Giraph) or other message-passing based frameworks (e.g. GraphLab [13]). Section 2 contains a brief introduction of the principle and implementation of G-Path. We believe G-Path will be useful for applications on social networks.

Also, we propose an interesting application for social web sites, which is built on G-Path. Our implementation of G-Path builds on Hadoop HDFS and Hama. The system takes user's search terms and then displays the query result in an interactive user interface. The dataset contains various kinds of vertices and edges to show the flexibility of G-Path queries. Section 3 introduces the demonstration system with some screenshots.

The following contributions are made in this paper:

a) We introduce a generic path pattern query language, called G-Path. A G-Path query can be compiled into a finite automaton and takes several optimization methods.

WWW 2013 Companion, May 13–17, 2013, Rio de Janeiro, Brazil.
ACM 978-1-4503-2038-2/13/05.

[1] http://www.neo4j.org/

[2] http://giraph.apache.org/

State	Condition			
	Predicate	Transition		
		Type	Predicate	To-State
1	id=1	Out	*	2
2	*	Accept	N/A	N/A

Table 1: An example of QFA table

b) A BSP-based parallel algorithm is introduced to process G-Path queries, which can easily integrate into existing graph-processing frameworks.

c) We have developed a demo application which performs a search on people and publications on a DBLP dataset. Users can use an interactive interface to create queries. The result is shown in a visual graph.

2. THE QUERY SYSTEM

The G-Path query system is responsible for searching for a given regular path pattern (introduced in Section 1) on a graph dataset. First, we will introduce the syntax of G-Path query language in Section 2.1.

G-Path query system consists of 3 parts: (1) Query compiler, which converts a literal query into an execution plan and optimizes the execution plan. In our system, an execution plan is represented by a finite automaton, called QFA. Section 2.2 introduces the definition of the QFA. (2) Query processor, which executes the finite automaton under bulk synchronization parallel model. Section 2.3 introduces the execution of a QFA and Section 2.4 gives some optimization strategies in brief. (3) Graph data manager, which reads and writes graph datasets on a HDFS cluster. This part is a lower level service of our system, so in this paper we do not describe this part in detail.

2.1 G-Path Query Language

G-Path query language is used to define a regular path pattern. It has a simple syntax but is general enough to describe different kinds of patterns.

The language has only 2 basic characters: "." (dot) and "-" (minus). A dot represents a vertex in the path while a minus represents a directed edge. For example, ".-." is a path with two vertices with an out-edge of the first vertex pointing to the second one.

G-Path language supports many regular expression operators. Such as: (1) alternation sign "|" which matches either its left part or right part; (2) quantifiers "*", "+" and "?" which means "zero or more times", "one or more times", "zero or once", respectively; (3) group "()" which packs a sub-query to a single matching unit.

Because G-Path is a query language on attributed graphs, we define a syntax for searching attributes, which is "[attr OP value]" filter. OP can be binary relational operators such as "=", "!=", ">", ">=", "<" or "<=". ">>" is another special operator which is only applied on string literals and matches only when the attribute's value contains the test string literal. Multiple attribute filters represent a conjunction of different constraints. For example, ".[year=2012] [name>>World]" matches a node with its "year" attribute equaling to 2012 and "name" attribute containing the word "World".

A legal path is a sequence in which vertices and edges alternate and the first as well as the last one in the sequence should be

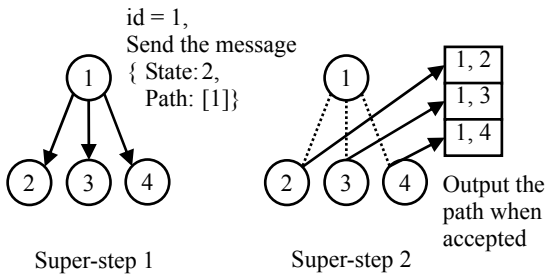

Figure 1: An example of execution steps

vertices. However, some queries violate this constraint, e.g., ".." has a concatenation of two vertices. Looking at this query, it is easy to infer that user wants to query for ".-.". G-Path language supports to omit some unrestricted nodes or edges, but omitting two concatenating entities is not allowed. We can infer omitted parts from the integrity constraint. For example, a query should begin with a vertex. So given a query "-", we must add a leading and a tailing "." to form a legal query ".-.", otherwise the integrity constraint is violated.

2.2 Query Finite Automaton

A valid query will be compiled to a query finite automaton (called QFA). Similar to a normal definite finite automation (DFA), a QFA consists of several states and transitions between states. However, the QFA is different from DFA.

Each state of QFA responses to a vertex in the path. Each transition between states increases the length of the path by 1. An obvious difference between QFA and DFA is that a QFA's state contains predicates on the vertex along with predicates on the vertex' edges. As a result, the QFA fits graph data better than normal DFA does, and can be easily executed by a BSP-based parallel executor.

A QFA can be represented with a table. Table 1 shows an example of query ".[id=1]-." which finds all sibling vertices of a given vertex. From the table we can see that the "transition" part of QFA is more complex than a DFA's. There are 3 types of transitions. "In" and "Out" correspond to a vertex's incoming edges and outgoing edges. "Accept" is a special type, which means that the automaton has already matched a desired path.

2.3 BSP-based Query Processing

A big advantage of our implementation of G-Path is that the query can be executed in parallel. We use a BSP-based query engine. Each BSP message (*State*, *Path*) can be treated as an instance of the QFA in which *State* is the current state number and *Path* is the matched path fragment. The QFA table is the same for all messages, so it does not need to store the table in messages.

For each received message in a super-step, the message can be treated as a QFA. Then we can calculate the next state along each edge for the current vertex. If the next state is valid, a message with the new state and path will be sent along the edge. Multiple messages are sent if more than one edge has a valid transition, which "forks" the execution of the QFA. This is the primary cause of the *parallel* execution. Figure 1 is an example of executing the QFA shown in Table 1. In the first super-step, vertex 1 sends a message of state 2 to its neighbors. In the second super-step, vertices 2, 3 and 4 each receives a message from vertex 1. Because state 2's transitions contain an *Accept*, a valid path will

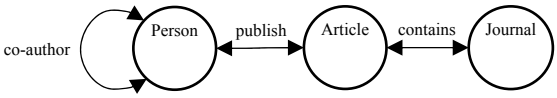

Figure 2: Data schema of demostration's dataset

be the output for each of them. The final result set is [1, 2], [1, 3] and [1, 4].

2.4 Optimization

Optimization is an important part of a practical data management system. We have developed several optimization mechanisms to process G-Path queries faster. These optimizations come from two aspects: QFA optimization and runtime optimization. In the following paragraphs we will introduce the most significant optimization of the two aspects.

Because the QFA supports different types of transitions, we can build several equivalent QFAs. For example, for query ".-.[id=1]", we can start in all vertices and find their outgoing vertices in which its ID is 1. An alternative way is to first find the vertex whose ID equals to 1, and then find all of its incoming vertices. The second way makes fewer messages than the first way. QFA optimization aims to find an optimal automaton in all equivalent variants.

We notice that for the same state and vertex, the following execution steps are almost identical except for matched paths. If we combine all messages of the same state, we can reduce a large amount of messages. Based on this observation, we develop tree compact optimization which compacts all messages in the same state and combines their matched paths into a tree. So a tree compact message can contain many original messages, reducing a large amount of messages.

3. THE DEMOSTRATION

Our demo is a web application built on the G-Path query system in which the DBLP dataset is used. In other papers, the DBLP dataset is often treated as a co-authorship network, which ignores most information about publications in the dataset. However, we preserve the information about articles and journals in DBLP to handle flexible queries.

Each vertex in the dataset contains an attribute "type", which indicates the type of the vertex. There are 3 types of vertices: *Person*, *Article* and *Journal*. All edges are undirected (thus can be retrieved in both directions) with different types and attributes. A *Publish* edge lies between *Person* and *Article* while a *Contains* edge lies between *Journal* and *Article*. If two people have a common article, a *Co-author* edge will connect them. Although *Co-author* edges can be inferred from *Publish* edges and *Article* vertices, we still preserve them for convenience. Figure 2 is the graphical data schema of the dataset. It shows different types of vertices and different types of edges between vertices. G-Path supports directed edges only, so each undirected edge is represented by two directed edges with same attributes.

The dataset contains 1,617,172 vertices and 6,323,177 edges. There are 713,124 different people, 902,746 articles and 1302 journals. The number of "co-author", "contains" and "publish" relationships are 3,095,497, 902,523 and 2,325,157, respectively.

With different types of vertices and edges, we can construct complex queries on different types and attributes. Section 3.1 gives an introduction to the demonstration application. Also, Section 3.2 gives some examples and explanations of the queries.

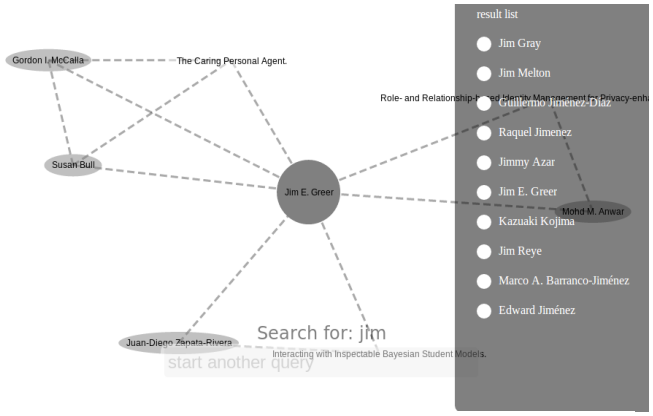

Figure 3: Finding a single person

3.1 User Interface

In the middle of the screen there are a textbox to put in queries and a big node (called the center node). Different center node can be chosen from the pull-down menu on the right. Direct G-Path queries are acceptable but not friendly to end users. As that being the reason, the application also supports simple searching terms as other searching engine does. The terms will be converted to queries on people, articles or journals.

The system requires all queries to start with a single vertex for a better display scheme (it is not a restriction of G-Path but only a restriction of the demo application). The result of the query is shown as a network graph. HTML5 techniques are used to develop a fancy interface.

3.2 Queries

In this section, we will give several types of queries that our system handles.

Finding a single entity can be done by inputting a name in the search box. It will be converted to a G-Path query like ".[name>>KEYWORD]" and *KEYWORD* represents the keyword of the query. By default, the system will search entities containing the keyword from all of the people, articles and journals. Specially, the *KEYWORD* with a leading @ indicates that the query is to find a person, the bracketed *KEYWORD* (i.e., *<KEYWORD>*) indicates to find a journal, and the quoted *KEYWORD* (i.e., *"KEYWORD"*) indicates to find an article. The result will be shown in the center node as in Figure 3. Note that the dashed lines are for the sake of presenting the neighbor nodes,

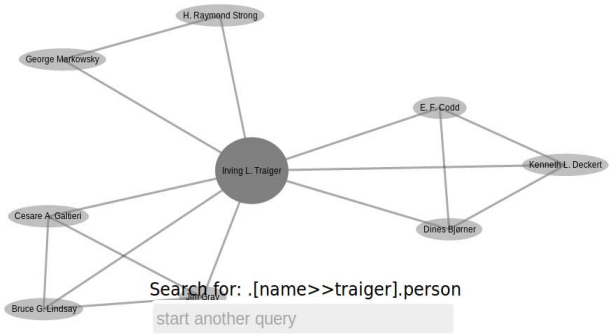

Figure 4: Co-author cluster

Figure 5: Finding similar people

which in fact are not included in the result (the result only contains a single person).

Finding co-author clusters aims at finding all co-authored people and their relationships of a given person. The G-Path query should be ".[name>>KEYWORD].[type=Person]". In this query, the first dot matches the given person and the second dot matched his co-author.

Figure 4 shows a result of this kind of query. You can see 3 cliques of the given person. Their dimensions are 3, 4 and 4. The right side of this figure is the detailed information of a selected person (connected with a dashed line).

Finding similar journals is another kind of query. It shows the flexibility of the system where users can not only find relationships between people, but also search according to relationships between journals. If a person published several articles in different journals, the journals may share a common interest of that person. It is reasonable to say that the journals are likely of similar interest.

Figure 5 shows the result of similar journal queries. Two different journals are "IBM Research Report" and "IBM Research Report, San Rose, California". Actually, they are referring to the same journal. However, because our DBLP dataset distinguishes journals only by their full names, the two journals are regarded as different journals. In the query's view, those two journals share the same interest and should be very similar.

Finding similar people shows that the system handles different kinds of relationships between people. A co-author relationship is a strong one, but sometimes we want to deal with weaker relationships. For example, if two people have published articles in a same journal, they may have a weak relationship (i.e., share a common interest). Figure 6 shows a result graph of this kind of queries, in which **circles** represent persons, **small rectangles** represent articles and **big rounded rectangles** represent journals.

4. CONCLUSION

In this paper we present G-Path, a flexible path query language along with a processing algorithm based on BSP parallel model. G-Path query language's syntax is simple but powerful. The algorithm is designed to be parallel so that it handles large graph data and can execute efficiently in a distributed computing cluster. Based on BSP parallel model, which is used by most common graph processing frameworks, G-Path can also be integrated into existing graph data management systems easily and improve the existing solutions.

A prototype of G-Path is developed, on which a web application is built with the DBLP dataset. Different from most other DBLP-based applications which use DBLP as a single co-authorship network, our system stores more detailed topology structure and attribute data of DBLP, which supports more flexible and complex queries.

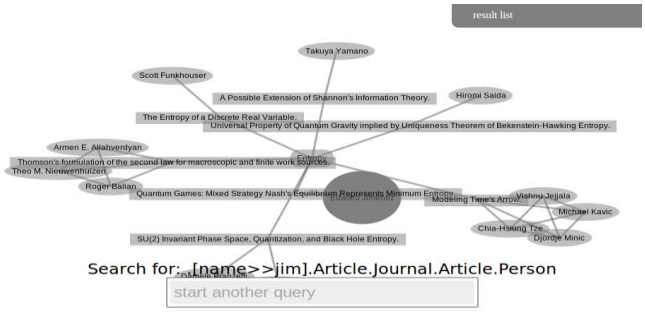

Figure 6: Finding similar journals

5. ACKNOWLEDGEMENTS

This work was supported by the National Natural Science Foundation of China (No. 61170064) and the National High Technology Research and Development Program of China (No. 2013AA013204).

6. REFERENCES

[1] W. Fan, J. Li, S. Ma, N. Tang and Y. Wu, "Adding Regular Expressions to Graph Reachability and Pattern Queries," *Frontiers of Computer Science,* vol. 6(3), pp. 313-338, 2012.

[2] L. Chen, A. Gupta and M. E. Kurul, "Stack-based Algorithms for Pattern Matching on DAGs," *VLDB,* pp. 493-504, 2005.

[3] R. Jin, Y.Xiang, N. Ruan and H. Wang, "Efficiently Answering Reachability Queries on Very Large Directed Graphs," *SIGMOD,* pp. 595-608, 2008.

[4] R. Giugno and D. Shasha, "Graphgrep: A Fast and Universal Method for Querying Graphs," *ICPR,* vol. 2, pp. 112-115, 2012.

[5] H. Kashima, K. Tsuda and A. Inokuchi, "Marginalized Kernels between Labeled Graphs," *ICML,* vol. 20(1), p. 321, 2003.

[6] W. Fan, "Graph Pattern Matching Revised for Social Network Analysis," *ICDT,* pp. 8-12, 2012.

[7] H. He and A. K. Singh, "GraphQL: Query Language and Access Methods for Graph Databases," Technical Report, University of California, Santa Barbara, 2007.

[8] R. Ronen and O. Shmueli, "SoQL: A Language for Querying and Creating Data in Social Networks," *ICDE,* p. 1595–1602, 2009.

[9] L. T. Detwiler, D. Suciu and J. F. Brinkley, "Regular Paths in SPARQL: Querying the NCI Thesaurus," *AMIA Annual Symposium Proceedings,* p. 161, 2008.

[10] E. Prud'hommeaux and A. Seaborne, "SPARQL Query Language for RDF," *W3C Recommendation,* vol. 15, 2008.

[11] L. G. Valiant, "A bridging model for parallel computation," *Communications of the ACM,* vol. 33(8), p. 103–111, 1990.

[12] G. Malewicz and M. H. Austern, "Pregel: a system for large-scale graph processing," *Proceedings of SIGMOD,* pp. 135-146, 2010.

[13] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola and J. M. Hellerstein, "Distributed GraphLab: a framework for machine learning and data mining in the cloud," *Proceedings of the VLDB,* vol. 5(8), pp. 716-727, 2012.