Enabling Ad Hoc Queries over Low-Level Geospatial Datasets

David Chiu and Gagan Agrawal
Department of Computer Science and Engineering
The Ohio State University
Columbus, OH 43210
{chiud,agrawal}@cse.ohio-state.edu

ABSTRACT

Technological advancements have enabled many applications to generate large amounts of data from scientific simulations and experiments at an astounding pace. In enabling sharing and effective use of this data for various classes of users, especially within the web 2.0, it is important to be able to support simple and intuitive interfaces. This paper describes a system that enables high-level queries over low-level geospatial datasets. Our technique involves a combination of natural language processing, machine interpretable metadata, and a service composition engine which dynamically constructs workflows for answering queries based on service and dataset availability. A specific contribution of this work is the *data-driven* capability in which we provide a framework to capture and utilize information redundancy that is present in heterogeneous geospatial data sources. Our approach does not require a standarized format for storing all data, or the implementation of a complex mediator-based querying framework. We have evaluated our system using several queries from the geospatial domain. Our results show that our query planning (workflow composition) time is negligible as compared to the query execution time. Furthermore, the workflow composition time stay linear with both the number of datasets and the total number of workflows.

1. INTRODUCTION

We have been observing an increasing amount of scientific data generated by cutting edge applications and simulations. This data can be collected using different measurement units, different reference coordinates, and stored in a variety of formats. This data could be extremely valuable for both researchers and practioners. However, querying such data poses several challenges: naïve users must interpret and interact with potentially complex queries to retrieve data, which is often returned in its original cryptic low-level format. Domain experts, on the other hand, are hindered by the tedious workflow of manipulating multiple Web forms in order to retrieve different types of data. In enabling the sharing of this data for various classes of users, especially within the emerging web 2.0 it is important to be able to support simple and intuitive interfaces. This paper describes such a system, which is driven by the geospatial domain.

In the geospatial community, data is gathered daily through such devices as on-site sensors and satellites. Various classes of users, ranging from those interested in purchasing a waterfront property to researchers trying to understand underlying phenomenon, are interested in datasets for different time periods and across disparate geographical regions. While traditional database integration approaches like the use of federated databases [33] or mediator-based systems [17, 34] can be applied, we are interested in solutions which require a significantly lower amount of effort. At the same time, we need to be able to address the challenges arising due to the characteristics of scientific data. As observed in geospatial datasets, these characteristics include, but certainly not restricted to:

- *Massive Volumes* — data may be collected in a continuous manner, e.g., gauge stations situated on coastlines transfer readings every few minutes.
- *Low-level Format* — data is normally stored in native low-level format, rather than in structured databases, and thus a standard method for data interaction is lacking.
- *Heterogeneous Data Sources* — effective query results might involve an integration of disparate data sources.
- *Temporal-Spatio Domain* — since geographical data is highly volatile, rigorous maintenance of descriptors such as location and date are imperative to providing accurate information. For instance, the shoreline of Hawaii today is much different than what it was a decade prior.
- *Deep Web data* — in many cases, datasets are normally stored as flat files in back-end file systems and accessed via specific queries and user interaction with Web forms. The National Oceanic and Atmospheric Administration (NOAA [26]) provides one such portal.

We have developed a system that enables high-level queries over low-level geospatial datasets. Our technique involves a

combination of natural language processing, machine interpretable metadata, and a service composition engine which dynamically constructs workflows for answering queries based on service and dataset availability. A specific contribution of this work is the *data-driven* capability in which we provide a framework to capture and utilize information redundancy that is present in heterogeneous geospatial data sources. Because new data sources and services may be introduced often in a distributed/shared GIS environment, such heuristics towards dynamic service composition might therefore help alleviate some hardships by actively searching for novel ways to approach queries.

Particularly, our approach is driven by the following observations. First, there is a growing trend towards *metadata standards* in various scientific domains, including the geospatial domain. As an example, CSDGM (Content Standard for Digital Geospatial Metadata) has been developed by FGDC (Federal Geographic Data Committee)[15]. The second observation is that sharing of tools and programs as Web services is becoming popular. A considerable movement towards integrating geographical data with the Web in both industry [18, 23, 19, 40] and academia [24, 6, 37, 11, 3] can be seen in recent developments.

Much of the success that Web services have achieved can be attributed to its platform independent protocol which provides a simple communications medium across heterogeneous systems. In addition to enabling interoperation between cross-platform machines, services enable distributed access to code, datasets, devices, etc., making them available to a broad range of users. Aside from the applications that individual services can already provide, support for more complex applications, e.g., integrated business processes and large scale scientific analysis, may benefit from a collaborative execution of services into the form of workflows. This process is generally known as *workflow* or *service composition*, an area of research that has received significant attention.

There exist several methods of service composition, most notably, static, user-guided, and dynamic (automatic). *Static service composition* is a process where specific workflows resident in a system are preprogrammed. Although static workflows are efficient for handling certain requests, the system can only handle a small set of specialized queries. Furthermore, addition, modification, or unavailability of services and datasets may require schedules to be reprogrammed. *User-guided service composition* seeks to involve users by allowing them to manage the construction of a workflow to answer their specified query. This is typically done by offering users an interface that features some graphical "building block" construction of workflows. To better guide the user, the usable set of building blocks goes through a semantic filtering process each time a block is set in place by the user. While these systems offer customization, the composition process still involves the identification of the correct services and datasets, which may seem daunting to non-experts whom might only be interested in casual queries.

The goal of *dynamic workflow composition* is to maximize transparency by eliminating intermediate user intervention altogether. Systems that employ dynamic workflow composition maintain an index and metadata of the available services by using some service discovery or registration method. Upon query requests, it generates execution paths on-demand through an exhaustive semantic and schema matching heuristic. Compared to the static approach, this is expectedly more time consuming than simply executing a prescheduled sequence, however, it offers multiple execution paths to provide more flexibility when approached with the aforementioned adversities. Against the user-guided approach, it may lack the personalization of the exact workflow, but saves the user from its construction.

We believe that dynamic workflow composition can be combined with machine-interpretable metadata, a domain ontology, and a natural language interface to offer simple and intuitive tools for querying a variety of scientific datasets, which are stored in low-level formats. Our approach does not require a standarized format for storing all data, or the implementation of a complex mediator-based querying framework. Besides expecting the availability of tools and programs as Web services, our approach requires that all data be annotated with a standard metadata.

Our system was evaluated using several queries from the geospatial domain. Results show that our query planning (workflow composition) time is negligible as compared to the query execution time. Furthermore, the workflow composition time stay linear with both the number of datasets and the total number of workflows.

The remainder of this paper is organized as follows. An overview of our system is presented in Section 2. In Section 3 we discuss technical details of the system. An evaluation of the system is given in Section 4. We compare our work with related research efforts in Section 5, and finally, we conclude and discuss future directions in Section 6.

2. SYSTEM OVERVIEW

A conceptual view of our system is shown in Figure 1. We present a quick overview of each component's requirements.

Query Parser — Among our system's goals, one is to provide some provision of user-friendliness. To this end, we want to support high-level user queries, which implies that a somewhat sophisticated natural language parser should be included. Specifically, the query parser takes a query and parses its relevant portions into concepts in our scientific domain.

Workflow Construction Engine — Given that a well-defined query, appropriate services and datasets are selected for use and their composition is reified dynamically through consultation with the ontological data. This component outputs a *set* of possible orderings on a sequence of services. Correctness is imperative in this feature, that is, all generated sequences of service execution must produce the targeted result. Discussed next, enriching available datasets and services by describing their interrelationships, along with relevant domain information, enables semantics matching to help guarantee workflow correctness.

Services and Data Annotations — Semantic descriptions of the available data and services including their interrelations

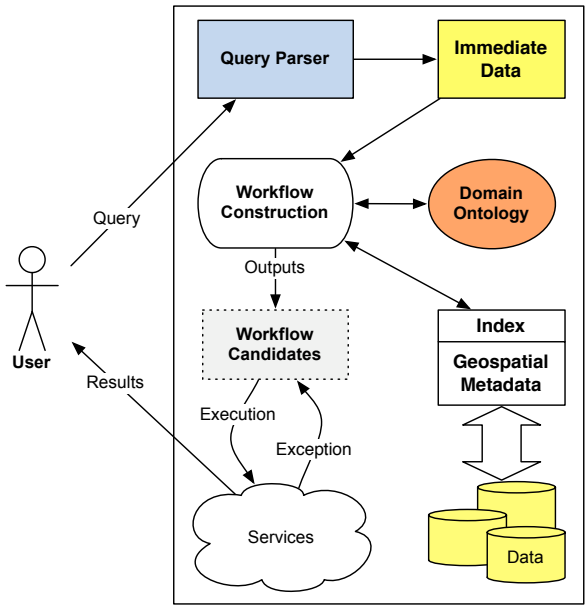


Figure 1: Overview of the System

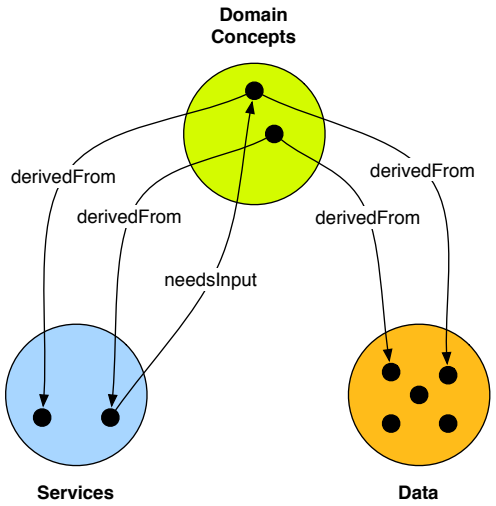


Figure 2: Ontology for Domain Specific Semantic Description

must be provided to the system. First, for geospatial datasets, metadata is specified in CSDGM (Content Standard for Digital Geospatial Metadata), as required by the Federal Geographic Data Committee (FGDC) [15]. CSDGM annotates data files with such descriptions as coverage, date of creation, coordinate system, etc. For services, their interface is typically described in WSDL. In addition to data type descriptions, our annotations also include domain information that will aid in supporting automatic determination of correctness such as dependencies and context suitability. Effective classification of geographic datasets and services along with a description of their interrelationships will help filter the set of services to those suitable for execution. We use a standard ontological descriptor, Web Ontology Language (OWL) [8], for this purpose.

3. TECHNICAL DETAILS

For clarity, we feel that it is necessary to first understand how the system captures the available data, services, and their methods of interoperation. We propose an ontology, depicted in its general form in Figure 2, which consists of three classes:

1. *Domain Concept Class* — Domain entities such as location, shoreline, parcel, etc., are classified as instances of this class.

2. *Data Class* — Data instances are represented as a pair containing the URLs to its content and metadata description.

3. *Service Class* — Each service instance encapsulates its WSDL file location and other semantic information such as preconditions on its parameters.

Having defined the top-level classes in our ontology, the next issue involves defining how each entity relates to one another. This step is imperative to our workflow construction engine, since our methodology for ensuring the workflow's semantic correctness is based directly on the decomposition of domain concepts into services and datasets. We designed two relationships:

1. *concept derivedFrom (service or data)* — This relationship states that concept instances contain outgoing *derivedFrom* links to either data or service instances. This denotes that high-level domain concepts can be somehow derived by some types of data or is the product of some service. Each concept must contain one or more *derivedFrom* links, or else there is no reason for the concept to exist in the system. For example, if there does not exist any sequence of services or data that can somehow return results on the concept of "wind," then the system cannot answer any queries involving wind.

2. *service needsInput concept* — Instances in the service class may contain zero or more *needsInput* links back to domain concepts. This relationship is necessary for the system to understand each service parameter's meaning. For example, assume some service S maintains two parameters, a and b. While it is simple to deduce (perhaps from S's WSDL description) the data types of both parameters, without domain information, it becomes intractable to determine just *what* they represent.

The ontology's design, admittedly simple, serves a purpose for future developments. While this paper specifically involves the geospatial domain, the ontology's open structure allows us to believe that it can be easily generalized to many other scientific domains. Since it is the semantic core of our

system, it is realistic to assume the possibility of our system supporting "plug-and-play" domains without involving too much effort.

We lead into the discussion of the technical specifications of each system component through a simple working example. For the remainder of this section, let's assume that the ontology as depicted in Figure 3 is given to the system. The first item to note is that it has been "unrolled" from the general form for visualization purposes. The nodes labeled C, S, and D correspond to concept, service, and data instances respectively. From the top, the waterLevel concept contains three service methods of derivation: getWL, WLFromALT, and retrieveData. Each of these service instances contains links back to domain concepts. For instance, retrieveData requires two parameters, whose concepts are coord and waterGauge. Focusing on the coord path, the system can understand that it can be derived by either immediate data (exact coordinates are given by user) or a service call to location2coord, which eventually sinks into an immediate data instance where the user must at least provide the location in the query.

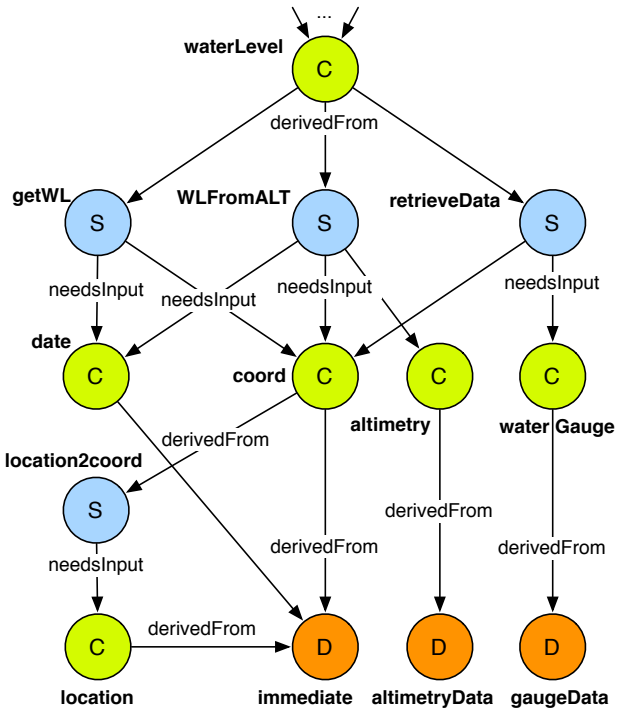


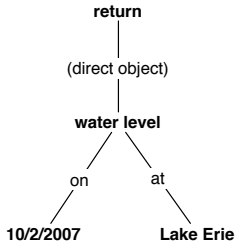
Figure 3: Example Ontology

The coord concept's subgraph itself exposes the much desired feature that *multiple* workflows can be generated to answer the same query. We mentioned earlier in the introduction that one of our goals is the provisioning of an intuitive user interface for all classes of users. Here is one such example. Whereas the naïve user might only supply a general location to the query, the domain expert may be able to provide the exact coordinates. Although one more service call is needed for conversion of a general location to approximate coordinates, the transparency of this process is the ultimate objective. Having described the example ontology, for the remainder of this section, assume that the following query has been submitted to the system:

```
''return water level at Lake Erie on 10/2/2007''
```

3.1 Query Parser

The objective of the natural language parser is to extract and attach domain information to the user given data. We want to not only uncover the final target concept of this query (water level), but also correctly decompose the user given data into the following *concept=value* pairs: *loc=Lake Erie*, *date=10/2/2007*. Using Stanford NLP [20] the query is broken into distinct grammatical relations as shown in Figure 4(a). Each relation represents a candidate of providing relevant data towards a machine interpretable query structure. To reify each relation into a specific domain concept, we perform synonym matching using WordNet [14] to look for materializations of domain concepts. Together with the semantics of each grammatical relation, we can expect the extracted immediate values in Figure 4(b).



(a) Query Parse Tree

	Concept
Query Target	*waterlevel*

Immediate Data = *imm[...]* =

$$\begin{cases} loc \rightarrow \{LakeErie\} \\ date \rightarrow \{10/2/2007\} \end{cases}$$

(b) Immediate Data

Figure 4: Query Parsing Process

These pairs of user-provided (immediate) data are stored in a hashtable $imm[\dots]$ and input into the service composition engine along with the target domain concept *waterLevel*. This table is keyed by concept instances, and each key points to linked list of given values that has been parsed under that concept.

Next, we define the workflow composition problem, and describe in-depth our solution.

3.2 Problem Statement

Under our framework, a workflow w is recursively defined as

$$w = \begin{cases} d \\ (s, P_s) \end{cases}$$

such that d is a data instance and (s, P_s) is a tuple where s denotes a service and parameter list P_s is an n-tuple $(p_1, \dots, p_n)$ and each p_i is a sub-workflow. In other words, a workflow is either a single data element or a service whose parameters are themselves products of workflows. Presumably, the execution of a workflow reduces it to a base data element, which is ultimately its final product. A workflow can also be represented as its (possibly) more familiar form of a *directed acyclic graph* (DAG) if we let vertices denote services and data elements, and directed edges denote the flow of data.

The best-effort aspect of this problem is akin to most dynamically natured systems in that upon faults it persistently attempts alternative, albeit potentially less optimal or approximate, solutions. To this end, we propose an algorithm that enumerates all workflow candidates from the given resources (services and data) in our system. The system then executes workflows within this set, iterating through candidates as needed.

3.3 Workflow Enumeration Algorithm

Domain concept derivation is the basic goal behind constructing each workflow. The algorithm takes a target concept, in our case $target = waterLevel$, and the set of immediate data, $imm[\dots]$, as input and outputs a set W of unique workflows that are capable of returning the desiderata for the target concept. From target concept, we simply traverse every outgoing edge until we reach a data node — recall from the problem statement that every well-defined workflow must somehow default to a data component, and thus, all sink nodes in the ontology are base data elements. Each workflows is built recursively from the bottom up with each split at a concept node defining a unique path (a new workflow) towards solving the target concept. This procedure, detailed in Algorithm 1, is akin to depth-first search on the ontology.

Returning to Figure 3, we can see that the waterLevel concept contains *at least* 3 workflows, as it is directly derivable by three types of services/data. In this case, waterLevel is derivable by three distinct services, i.e., (Line 7) assigns $B \leftarrow \{getWL, WLFromALT, retrieveData\}$. Without loss of generality we focus simply on the getWL path. This particular service consists of two parameter concepts: (Line 21) of the algorithm thus assigns P with two parameters of concept types *date* and *coord* respectively.

(Lines 22-31) build a running set, Δ, of sub-workflows that can be used derive all parameters in P by calling the algorithm recursively on each parameter's concept. That is, each $\delta \in \Delta$ denotes a set of candidate sub-workflows that derives each corresponding parametric concept. Notice that a recursive call is avoided through memoization (Lines 24-25) if the concept node has already been visited and solved. Otherwise, we are forced to make a recursive call (Lines 28-29) on the given concept node with a potentially narrowed set of immediate data. The motivation behind redistributing the set of immediate data will be discussed later.

To further our example, Δ is built by making consecutive recursive calls on *date* and *coord*. Here, *date* has a single path of derivation through immediate data while *coord* can be derived through both immediate data or calling the loc2coord service, and Δ is thus assigned $\{\{(imm[date])\}, \{(imm[coord]), (loc2coord, (imm[loc]))\}\}$.

In substantiating the service parameter list we must account for all combinations of the sub-workflows contained in Δ. We compute the cross product of all $\delta \in \Delta$ (Line 32) to obtain a set of distinct parameter lists $Params = \{(imm[date], imm[coord]), (imm[date], loc2coord(imm[loc]))\}$. Each element in $Params$ is then coupled with the original service, getWL, to obtain the set of unique workflows. The process continues for the remaining services used to derive the original concept to obtain the set W of candidate workflows, shown in Table 1. This example extracts w_3 and w_4.

Algorithm 1 enumWF($target$, $imm[\dots]$)

```
 1: W ← ∅
 2: /* static arrays for memoization */
 3: global visited[...]
 4: global subWorkflows[...]
 5:
 6: /* B denotes the set of all data/service elements that
    can be used to derive target concept */
 7: B ← derives(target)
 8: for all β ∈ B do
 9:   if β is a data element then
10:     if β is immediately available from query then
11:       W ← W∪ imm[target]
12:     else
13:       /* β denotes dataset */
14:       for all file ∈ getFiles(β, imm[...]) do
15:         W ← W ∪ file
16:       end for
17:     end if
18:   else
19:     /* β is a service element */
20:     /* P denotes the set of service params */
21:     P ← getServiceParams(β)
22:     Δ ← ∅
23:     for all p ∈ P do
24:       if visited[p.concept] = true then
25:         Δ ← Δ ∪ subWorkflows[p.concept]
26:       else
27:         /* redistributing immediate data of each
            param may be necessary for correctness */
28:         imm'[...] ← redistribute(β, p, imm[...])
29:         Δ ← Δ ∪ enumWF(p.concept, imm'[...]))
30:       end if
31:     end for
32:     Params ← crossProduct(Δ)
33:     for all pm ∈ Params do
34:       W ← W ∪ (β, pm)
35:     end for
36:   end if
37: end for
38: visited[target] ← true
39: subWorkflows[target] ← W
40: return W
```

Of course, it is expected that not all candidates in W can be materialized due to a lack of provided immediate data. After all, it would make little sense for the user to provide both a location specific coordinates in the same query. Assuming that only the location is given, and each dataset element identifies only a single file url, then the true workflow candidates are uncovered, $W = \{w_2, w_4, w_6\}$.

3.4 Data Identification

One abstraction with data handling is the getFiles(...) subroutine on (Line 14) which crawls through the system's index on data files given the immediate data list (which contains geospatial properties as required by the user) and returns a set of URLs to the relevant data. In order for this routine to operate accurately, our system requires each file to

Table 1: Workflows for Deriving "Water Level"

W	
w_1	(retrieveData, (imm[coord], url[gaugeData]))
w_2	(retrieveData, (loc2coord, (imm[loc])), url[gaugeData])
w_3	(getWL, (imm[date], imm[coord]))
w_4	(getWL, (imm[date], (loc2coord, (imm[loc]))))
w_5	(WLfromALT, (imm[date], imm[coord], url[altimetryData]))
w_6	(WLfromALT, (imm[date], (loc2coord, (imm[loc])), url[altimetryData]))

be coupled with the federal standard metadata description, CSDGM, as mentioned in the previous section. CSDGM annotates each file with information such as its area of coverage and date of relevance.

Specific to the metadata standard, the date (or range of dates) is given by the CSDGM element `<timeinfo>`. In our datasets, each file contains a single date of relevance, e.g.,

```
<timeinfo>
    <sngdate>
        <caldate>20050708</caldate>
    </sngdate>
</timeinfo>
```

The spatial coverage, denoted in `<spdom>`, is given by using 4 boundary lines. The graphical representation of this information is provided in Figure 5:

```
<spdom>
    <bounding>
        <westbc>0</westbc>
        <eastbc>78.89</eastbc>
        <northbc>42.87</northbc>
        <southbc>0</southbc>
    </bounding>
</spdom>
```

The brute force solution for identifying the correct data files is to linearly match user requirements against each file's corresponding metadata description on its spatial coverage and time of relevance. Linear search, however, will undoubtedly dominate our algorithm's response time. Instead, we decided on a multilevel index on each file's spatial coverage and date. Although this multilevel index is enough for running experiments on our small experimental datasets, more sophisticated spatio-temporal indices [31, 4] should be used in practice due to the fine granularity of both properties.

3.5 Mapping Immediate Data

The other complication with data involves those that are user provided. Notice that the workflow enumeration algorithm does little to guarantee any form of correctness of its generated workflows. We use another application-driven example to expose this problem. Consider a service, getDiff(L_1, L_2), that returns the land surface change between two topological data files L_1 and L_2 (assume that

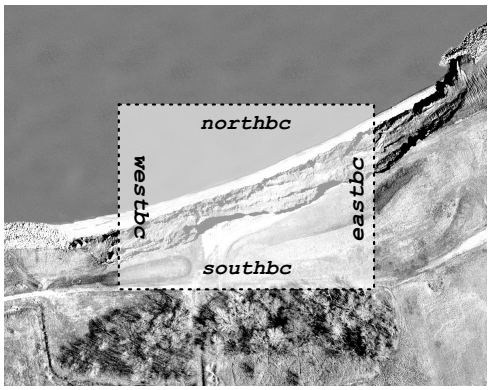


Figure 5: CSDGM Spatial Bounding Information

these files each contain a single matrix). Description of this service is simple. It assumes that L_1 is a dataset obtained at an ealier time than L_2, and that they both belong to the same geographical region; the service simply outputs $L_2 - L_1$.

In order for the service to make these assumptions, we must ensure that the data given as are reliable. We draw on well-established solutions for program correctness by requiring that a set of preconditions be met before allowing the execution to be deemed valid. It should be clear that the following precondition must be captured in order to implement the above semantics: $(L_1.date \leq L_2.date \wedge L_1.loc = L_2.loc)$. Staying in line with our example, assume that the query parser outputs the following immediate dataset:

$$imm[\dots] = \begin{cases} loc \rightarrow \{(x, y)\} \\ date \rightarrow \{10/2/2007, 12/3/2004\} \end{cases}$$

To satisfy our precondition, we distribute the values accordingly down their respective parameter paths. Called on (Line 28) of the algorithm, this procedure is illustrated in Figure 6. Notice that if no possible assignment exists, the workflow candidate is implicitly discarded by the recursive call in (Line 29) because imm' would not contain enough immediate data to substantiate the workflow.

3.6 Complexity Analysis

In terms of time complexity, recalling that our proposed ontology is a semantically glorified DAG, the enumeration algorithm is reducible to Depth-First Search (DFS). We can observe that, by initiating with the target concept node, it is necessary to traverse all intermediate nodes until we reach the sinks (data), leaving us with a number of distinct paths and giving our algorithm the same time complexity as DFS, $O(|E| + |V|)$. For clarity, we decompose its set of vertices into three familiar subsets: concepts nodes C, services nodes S, and data nodes D, i.e., $V = (C \cup D \cup S)$. Since the maximum number of edges in a DAG is $|E| = \frac{|V|*(|V|-1)}{2}$, our algorithm yields an $O((|C| + |D| + |S|)^2)$ *worst case* upper bound. Although theoretically sound, we argue that this measure is excessively conservative, and a recount of the

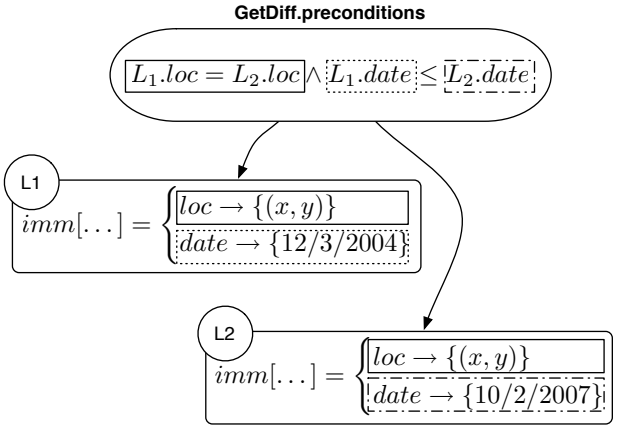


Figure 6: Redistribution of Immediate Values

defined structure of our ontology help justify this claim.

1. $(\nexists(u,v) : u,v \in K | K \in \{C,S,D\})$ No edges exist within its own subgraph.
2. $(\nexists(u,v) : u \in S \wedge v \in D)$ Edges from service to data nodes are not allowed.
3. $(\nexists(u,v) : u \in D)$ Data nodes are sinks, and thus contain no outgoing edges.

A more accurate measurement of the maximum number of edges in our ontology should be computed with the above constraints, in which we obtain $|E| = |C| * (|S| + \frac{|D|}{2})$, and thus giving us a significantly tighter upper bound.

4. EXPERIMENTAL RESULTS

The experiments that we conducted are geared towards exposing two particular aspects of our system. We show that our workflow enumeration algorithm is efficient compared to workflow execution time. We also present our system's capability for scaling to large amounts of available data and services. Our system was evaluated on a Linux machine running off Pentium 4 3.00Ghz Dual Core with 2GB of RAM. In order to somewhat stabilize workflow execution time all available data files that are required by queries are local, and the geospatial Web services are provided on a separate server located across our campus' network. While it is inconceivable to suggest that all indexed datasets reside on the local machine, we argue that local data access will only bias (by possibly shortening) workflow execution time, which is not the focus of these experiments.

Three application driven queries, as outlined in Table 2, were provided to us by domain experts at the Department of Geodetic Sciences here at Ohio State. While the type of queries that one can issue is innumerable, these were especially designed to capture our performance evaluation requirements while still pertaining to practical applications.

Our first query, Query 1, involves the interoperation of 4 distinct services to retrieve Deep Web data from NOAA.

Table 2: Experimental Queries

Query 1	"return water level of at (x, y) on 07/08/2004 at 06:18"
Query 2	"return surface topology difference at (x, y) from 07/08/2003 to 07/08/2005"
Query 3	"return shoreline extraction at (x, y) on 07/08/2004 at 06:18"

These services include (S_1) GetGSList: performs a lookup that retrieves a list of all currently available water gauge stations, (S_2) GetClosestGS: given coordinates and a list of gauge stations this service extracts the ID of closest gauge station, (S_3) GetWL: extracts the waterlevel at the given gauge station ID and date (returns a list of readings delimited by time), and (S_4) GetTimeWL: extracts the exact reading at the given time. Our system is actually capable of generating three correct workflows to solve this query:

- $w_1 = S_4(S_3(S_2(S_1())))$ — this workflow uses S_1 to fetch the gauge station list from Deep Web, which guarantees that the most up-to-date list of gauge stations is always used. This is significant since the result of S_2 directly relies on this list.
- $w_2 = S_4(S_3(S_2(DATA_{gaugeList})))$ — this workflow will search the system's data index for a cached list of gauge stations. This is useful if either S_1 or if the gauge station list residing in Deep Web is unavailable.
- $w_3 = S_4(S_3(imm[GSID]))$ — this workflow takes the gauge station ID directly from the user, saving 2 superfluous service calls. This one may be preferred by domain experts who are only interested in a particular gauge station's readings. Note that since $GSID$ is not given in the query, this particular workflow would not have been enumerated. However, for completeness, we decided to list its possibility.

The results for Query 1 are shown in Table 3. The execution times reported for each workflow is an average of 10 distinct runs, and as expected, are proportional to the number of service calls (although this may not always be true in general). Compared to the execution times, the runtime of our enumeration algorithm is diminutive. We believe that these results suggest that our system is capable of efficiently streamlining a potentially complex process through service composition that may have taken a user several steps to complete. Moreover, it has the option to dynamically attempt various workflow candidates upon initial faults/exceptions, and thereby supplying robustness to the user.

Table 3: Results for Query 1

Workflow Enumeration Time		
5.0×10^{-4} sec		
Workflow Execution Time		
w_1	w_2	w_3
2.401 sec	2.127 sec	1.234 sec

The second and third queries were designed to deal with geospatial datasets that are indexed on their temporal-spatio metadata information. Query 2 is interested in the surface difference between some location in a 2-year span. It involves a service, GetDiff, which is given two separate digital elevation model (DEM) files d_{old} and d_{new}. These files must be correctly identified using the given time and location. For correctness, our workflow construction engine differentiates these two parameters by distributing the dates correctly to its parameters using the methods discussed in Section 3.5. Query 3 extracts shoreline information by using a service GetShoreline, which takes as input the results of Query 1 combined with a DEM file pertaining to the same location. This is a two-step process; it must first construct the exact Deep Web extraction workflow as Query 1 to retrieve a water level value, and then correctly identify the resident DEM file.

Query 2 and 3 are used to expose system scalability. Because Query 1 does not involve data identification, it is not included in these evaluations. Note that the average workflow execution times for both queries are shown in Table 4. It is well-expected that the execution time for Query 3 will take much longer due to its Deep Web access, whereas Query 2 involves a single service invocation. In our first experiment, we synthetically generated an increasing number of DEM files that must be searched. For our experiments, the size of each file was 1 KB. The workflow enumeration times that are shown in Figure 7 suggest that the algorithm scales linearly against the varying amount of available data candidates. Furthermore, the line for Query 2 appears to dominate Query 3 because it needs to access an additional DEM file.

Table 4: Execution Times for Queries 2 and 3

	Query Execution Time
Query 2	0.772 sec
Query 3	2.41 sec

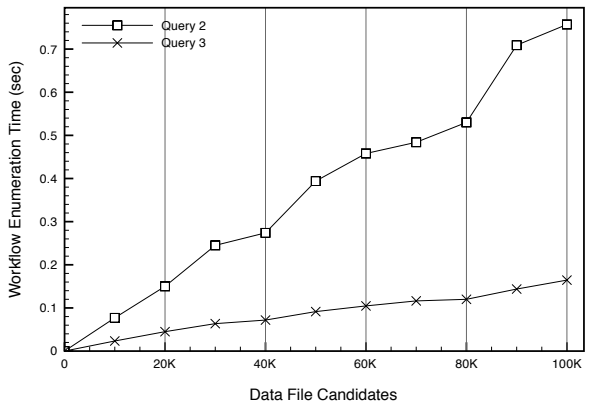


Figure 7: Scaling to Data File Candidates

Our next evaluation examines scaling to the number of workflow candidates, $|W|$. The results, depicted in Figure 8, show that the system again scales linearly to the size of workflow

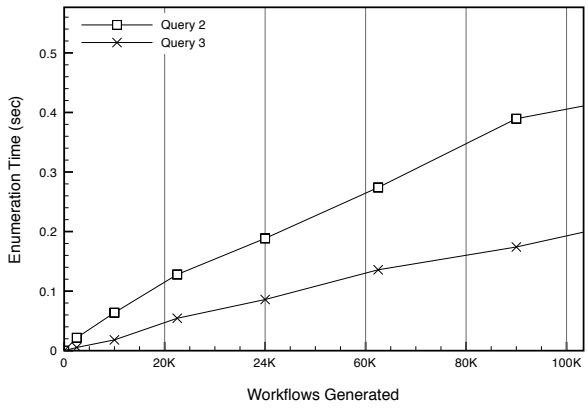


Figure 8: Scaling to Workflow Candidates

candidates. To obtain $|W| = |D| * |S|$, we fixed $|D| = |S|$ for $|S| = 50, 100, 150, 200$, etc. Admittedly, as $|W|$ gets very large, e.g., $|W| > 1M$, the system will begin to thrash since the amount of storage needed to store the candidates saturates system memory. But again, we stress that the expected size of $|W|$ in practice is very small compared to these extremities.

5. RELATED WORK

The call for a semantic-conscious Web has been addressed by such specifications as the Resource Description Framework (RDF) and its complement the Web Ontology Language (OWL) [22, 8]. These specifications allow for a standard framework for injecting metadata into any variety of applications. This supplies common machine-readable resources with a means of machine-understanding and interpretability. In our proposed system, we utilize RDF to formalize a general ontology which describes the relationships between certain domain-specific concepts and resources (datasets and services) available over the Web. This resource description is imperative to our system, as it semantically drives our workflow/service composition algorithm.

Service composition [13, 30, 27] is deeply rooted in legacy workflow management systems, where a streamlined execution of well-defined "processes" are used to define complex tasks in business and scientific operations. In fact, to complement the growing need for interoperability and data integration, many prominent workflow managers [9, 28, 2, 21] have evolved into service-oriented systems. These systems typically allow domain experts to define static workflows through a user-friendly interface, and map the component processes to known Web services. By itself, service composition have become prevalent enough to warrant such industrial standards as the WSBPEL (Web Service Business Process Execution Language) [38] to describe the orchestration of service execution. Implementations of WSBPEL engines have already sprawled into realms of proprietary and open-source communities, an auspicious indication of the high optimism for the movement towards composite service solutions.

Static composition systems (e.g., Microsoft BizTalk Server

[5]) are effective with the absence of changes in the computing environment such as the introduction or replacement of services. Such systems typically exist under proprietary domains, where the set of workflow processes and their components are rigorously defined and maintained. These systems, however, are inadequate in the face of a dynamic computing environment where new services are made available and reimplemented on a daily basis.

Many systems have been proposed to alleviate the painstaking task of maintaining consistency and correctness of the composite services under this environment. For instance, [7, 25, 12] describe a hybrid support for static workflows under dynamic environments. In HP's eFlow [7], a workflow's structure (known as a process schema) is first defined by some authorized users, but the instantiation of services within the process is dynamically allocated by the eFlow engine. It is also worth noting that eFlow also supports high level modification of schemas when necessary. Sirin et al. proposed a user interactive composer that provides semi-automatic composition [35]. In their system, after each time that a particular service is selected for use in the composition, the user is presented a filtered list of possible choices for the next step. This filtering process is made possible by associating semantic data with each service. SELF-SERV [32] is also user-guided but at a more abstract level where the actual instantiation of services is dynamically chosen. Traverso et al. discussed the importance of exploiting semantic and ontological information for automating service composition [36]. Their approach generates automata-based "plans", which can then be translated into WSBPEL processes. The goals and requirements for these plans, however, must be expressed in a formal language, which may be cryptic for the average user. Other planning-based systems [29, 39] also require similar complexity in expressing workflows. Fujii and Suda [16] proposed a 3-tier architecture for semantics-based dynamic service composition. The system uses natural language processor to parse queries into "components", and performs semantic matching to assure that a composed service satisfies the semantics of the query.

More pertinent to our work, the impetus and merits behind geospatial service composition have previously been highlighted in [1]. Di et al. described their system for ontology-based automatic service composition [10]. Here, workflows are composed via a rule-based system, and correctness is kept through backwards reasoning. However, base rules that are needed to generate a specific geospatial concept must explicitly defined in the system.

6. CONCLUSION AND FUTURE WORK

In this paper we have presented a system which supports simplified querying over low-level geospatial datasets. The entire process is enabled through a combination of effective indexing over metadata information, a system and domain specific ontology, and a workflow construction algorithm capable of alleviating all tiers of users of the difficulties one may experience through dealing with the complexities of scientific data. Our experiments have shown that the workflow construction algorithm is indeed capable of enumerating all possible composite services efficiently. Even under extreme circumstances of maintaining large numbers of data and workflow candidates, our results show that the enumeration time is insignificant compared to the actual workflow execution time.

It is difficult, nonetheless, to ignore the looming possibility of a large workflow candidate set. As we mentioned in the end of Section 4, the enumeration algorithm will not scale to massive amounts of workflows to answer one ad hoc query. While the paper addresses the possibility of efficiently enumerating all possible workflows, often in practice, it is an unnecessary and time consuming task. We are currently in collaboration with experts in the Department of Geodetic Sciences to implement a cost model that includes, among others, a predictive accuracy model for geospatial datasets. This accuracy model involves an acute understanding of each geo-data source and the possibility of propagation errors. Combined with a method that calculates network and computational latencies, our goal is to infuse this model with the enumeration algorithm to obtain a cost-effective heuristic for workflow optimality.

7. REFERENCES

[1] N. Alameh. Chaining geographic information web services. *IEEE Internet Computing*, 07(5):22–29, 2003.

[2] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludscher, and S. Mock. Kepler: An extensible system for design and execution of scientific workflows, 2004.

[3] E. Amitay, N. Har'El, R. Sivan, and A. Soffer. Web-a-where: geotagging web content. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 273–280, New York, NY, USA, 2004. ACM Press.

[4] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[5] Microsoft biztalk server, http://www.microsoft.com/biztalk.

[6] O. Buyukkokten, J. Cho, H. Garcia-Molina, L. Gravano, and N. Shivakumar. Exploiting geographical location information of web pages. In *WebDB (Informal Proceedings)*, pages 91–96, 1999.

[7] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M.-C. Shan. Adaptive and dynamic service composition in eFlow. In *Conference on Advanced Information Systems Engineering*, pages 13–31, 2000.

[8] M. Dean and G. Schreiber. Owl web ontology language reference. w3c recommendation, 2004.

[9] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. C. Laity, J. C. Jacob, and D. S. Katz. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[10] L. Di, P. Yue, W. Yang, G. Yu, P. Zhao, and Y. Wei. Ontology-supported automatic service chaining for geospatial knowledge discovery. In *Proceedings of American Society of Photogrammetry and Remote Sensing*, 2007.

[11] J. Ding, L. Gravano, and N. Shivakumar. Computing geographical scopes of web resources. In *VLDB '00:*

Proceedings of the 26th International Conference on Very Large Data Bases, pages 545–556, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[12] P. Doshi, R. Goodwin, R. Akkiraju, and K. Verma. Dynamic workflow composition using markov decision processes. In *ICWS '04: Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, page 576, Washington, DC, USA, 2004. IEEE Computer Society.

[13] S. Dustdar and W. Schreiner. A survey on web services composition. *International Journal of Web and Grid Services*, 1(1):1–30, 2005.

[14] C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. The MIT Press, 1998.

[15] Metadata ad hoc working group. content standard for digital geospatial metadata, 1998.

[16] K. Fujii and T. Suda. Semantics-based dynamic service composition. *IEEE Journal on Selected Areas in Communications (JSAC)*, 23(12), 2005.

[17] H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. Integrating and Accessing Heterogenous Information Sources in TSIMMIS. In *Proceedings of the AAAI Symposium on Information Gathering*, 1995.

[18] Google local, http://local.google.com.

[19] Google maps, http://maps.google.com.

[20] D. Klein and C. D. Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Meeting of the Association for Computational Linguistics*, pages 423–430, 2003.

[21] S. Majithia, M. S. Shields, I. J. Taylor, and I. Wang. Triana: A Graphical Web Service Composition and Execution Toolkit. In *Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, pages 514–524. IEEE Computer Society, 2004.

[22] F. Manola and E. Miller. Resource description framework (rdf) primer. w3c recommendation, 2004.

[23] Mapquest, http://www.mapquest.com.

[24] K. S. McCurley. Geospatial mapping and navigation of the web. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 221–229, New York, NY, USA, 2001. ACM Press.

[25] D. Mennie and B. Pagurek. An architecture to support dynamic composition of service components. In *Proceedings of the 5th International Workshop on Component -Oriented Programming*, 2000.

[26] National oceanic and atmospheric administration (noaa), http://www.noaa.gov.

[27] S.-C. Oh, D. Lee, and S. R. T. Kumara. A comparative illustration of ai planning-based web services composition. *SIGecom Exch.*, 5(5):1–10, 2006.

[28] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[29] S. R. Ponnekanti and A. Fox. Sword: A developer toolkit for web service composition. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, 2002.

[30] J. Rao and X. Su. A survey of automated web service composition methods. In *SWSWPC*, pages 43–54, 2004.

[31] T. K. Sellis, N. Roussopoulos, and C. Faloutsos. The r -tree: A dynamic index for multi-dimensional objects. In *The VLDB Journal*, pages 507–518, 1987.

[32] Q. Sheng, B. Benatallah, M. Dumas, and E. Mak. Self-serv: A platform for rapid composition of web services in a peer-to-peer environment. In *Demo Session of the 28th Intl. Conf. on Very Large Databases*, 2002.

[33] A. Sheth and J. Larson. Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, 1990.

[34] L. Shklar, A. Sheth, V. Kashyap, and K. Shah. InfoHarness: Use of Automatically Generated Metadata for Search and Retrieval of Heterogeneous Information. In *Proceedings of CAiSE*, 1995.

[35] E. Sirin, B. Parsia, and J. Hendler. Filtering and selecting semantic web services with interactive composition techniques. *IEEE Intelligent Systems*, 19(4):42–49, 2004.

[36] P. Traverso and M. Pistore. Automated composition of semantic web services into executable processes. In *3rd International Semantic Web Conference*, 2004.

[37] A. G. Woodruff and C. Plaunt. Gipsy: automated geographic indexing of text documents. *J. Am. Soc. Inf. Sci.*, 45(9):645–655, 1994.

[38] Web services business process execution language (wsbpel) 2.0, oasis standard.

[39] D. Wu, E. Sirin, J. Hendler, D. Nau, and B. Parsia. Automatic web services composition using shop2. In *ICAPS'03: International Conference on Automated Planning and Scheduling*, 2003.

[40] Yahoo! local maps, http://maps.yahoo.com.