Composing Geoinformatics Workflows with User Preferences

David Chiu  Sagar Deshpande‡  Gagan Agrawal  Rongxing Li‡
Department of Computer Science and Engineering
Department of Civil and Environmental Engineering and Geodetic Science
Ohio State University
Columbus, OH 43210

ABSTRACT
Service-oriented science, an interoperable paradigm for enabling computations over distributed heterogeneous systems, has not eluded the geographical community. Recent efforts toward standardizing geospatial technologies communicate this cause. Consequently, a number of quality systems have been built for cooperative geospatial data analysis and retrieval. Among these, workflow management systems have garnered considerable attention for automating and scheduling service executions. Traditionally, such systems seek to offer support for user preferences, e.g., time of completion. But with the rebirth of distributed computing towards the heterogeneous Data Grid, new challenges are posed for effectively supporting this feature: multiple disparate data sources, networks, compute nodes, and the potential for moving very large datasets. We believe that the relationship between execution time and workflow accuracy can be exploited to offer more flexibility in handling user preferences.

In this paper we discuss a system which enables user settings through a novel dynamic accuracy (sampling rate) adjustment scheme of workflow results and a framework for defining cost models for predicting application-specific completion times and the propagation of errors as an effect from sampling. We present our system in the context of two geospatial workflow applications. Experimental results show that accuracy adjustment resulted in a maximum deviation of 5.04% from the expected accuracies and an overhead on the order of \( \mu \text{secs} \). The effects of sampling on a particular workflow’s physical accuracy was also evaluated, displaying highly acceptable results even among diminutive sample sizes of the original dataset.

1. INTRODUCTION
The burgeoning advancements of web and grid technologies have helped launch a call for the availability and distribution of data and resources in various domains. Specifically within the geosciences, authorities, such as the Federal Geographic Data Committee (FGDC) and the Open Geospatial Consortium (OGC), have pushed this initiative through the standardization of geo-semantics and geospatial services respectively. This movement is a new direction for GIS applications which have historically been centralized on single machines. Recognizing the advantages of distributed heterogeneous data integration and interoperability, prominent proprietary applications, such as ESRI ArcGIS, have already integrated these standards into their frameworks.

With distributed heterogeneous datasets and processes comes the nontrivial challenge for scientists and other end-users to manage geoinformation. For instance, certain information involves execution of several processes with disparate data sources in a particular sequence, which may also involve any combination of data integration, cleansing, and other preprocessing tasks. Certainly, the ultimate hope for enabling these process sequences (traditionally known as service chains or geospatial workflows [2]) is to automate their composition while simultaneously hiding low-level details such as service and data discovery, integration, and scheduling from the user. Thus, many efforts in geospatial workflow management systems [17, 16, 28, 10] have been initiated to address these tasks.

But along with any user-oriented system is the need for supporting user preferences. Often, there are multiple ways of answering a given query, using different combinations of data sources and services. Some combinations are likely to result in higher cost, but better accuracy, whereas other options might lead to quicker results, but lower accuracy. This could be because some data collection methods involve higher resolution than others, or because some datasets are available at servers with lower-access latencies than others. Thus, query execution time is a natural user preference in any workflow system. Additionally, in many scientific disciplines, including geoinformatics, data reduction methods (e.g., feature selection, dimensionality reduction, . . .), can speed up the time for remote retrieval and computations, but will likely lower the accuracy. In the meantime different classes of users can have different querying requirements. Some users may want the answers the fastest, some may want the most accurate answers, and others might prefer the faster of the methods which can meet certain accuracy constraints. While most efforts in workflow management systems focus directly on minimizing execution times [25, 29, 1] through smart scheduling heuristics, it would be highly de-
sirable if we could enable user preferences for both accuracy and time. In other words, we seek to alleviate users from the need of understanding the cost and accuracy tradeoffs associated with different datasets and services that could be used to answer a query. This paper presents such a framework for geospatial workflow composition, which uses a novel approach for dynamically supporting user preferences on time and accuracy.

To automate the time/accuracy tradeoff in workflow management, we allow developers to expose an accuracy parameter, e.g., sampling rate. Our system also takes as input arbitrary models for predicting process completion time and error/accuracy propagation of the applications. We studied the effects of data sampling on the physical accuracy of the output from two specific geoinformatics applications: land elevation change and shoreline extraction, and extracted error propagation models from these methods. Our workflow composition algorithm employs an efficient algorithm to automatically regulate the accuracy parameter based on these cost predictions. An effective adjustment of this parameter can affect the overall execution time and accuracy of the composed workflows to meet user requirements.

We conducted experiments to evaluate three specific aspects of our system. First, we want to show that, although the prediction models are invoked quite frequently, they incur little contribution to the overall workflow composition time. Next, we measure the efficiency and efficacy of our accuracy parameter adjustment algorithm. Lastly, we discuss an evaluation of one error model’s prediction as compared to real physical error calculations. The remainder of this paper is organized as follows. An overview of our system is presented in Section 2. Error Prediction models for our considered geospatial queries are discussed in Section 3. In Section 4 we discuss technical details of our cost model and workflow composition algorithm. Performance evaluations of our workflow composition algorithm, error prediction models are presented in Section 5, and a comparison of our work with related research efforts follows in Section 6. Lastly, we conclude and discuss future opportunities in Section 7.

2. SYSTEM OVERVIEW

We now discuss a brief overview of each system component while directing the reader to detailed descriptions in our previous work [5]. A conceptual view of our system is shown in Figure 1. Most autonomous systems for information retrieval, including our own, require semantic descriptions of datasets and services. For geospatial datasets, CSDGM (Content Standard for Digital Geospatial Metadata) [12] is widely recommended for this purpose. CSDGM annotates data files with such descriptions as area coverage, date of creation, coordinate system, etc. Service interfaces are described in the WSDL standard [6]. In addition, we must also require that our semantic annotations include domain information that will aid in supporting autonomous determination of workflow correctness such as dependencies and context suitability. Effective classification of datasets and services along with a description of their relationships will help filter the set of services to those suitable for execution. The standard ontology descriptor, Web Ontology Language (OWL) [8], was used to define our simple domain ontology, shown in Figure 2, is not nearly as complete as other efforts in the geospatial domain [20], but it serves our specific purpose of workflow composition, and general enough to port to other domains. It allows developers to trivially define the relationships domain concepts, datasets, and services.

```
''crop an aerial image of Columbus, Ohio
```
In this study, we will discuss the two applications independently. The application queries, predictions, we must analyze the effect of data sampling with respect to physical error propagation. The application queries, which results in different errors along the horizontal and vertical directions.

### 3. ERROR PREDICTION MODELS

In our considered geospatial workflows, we focused on the particular adjustable accuracy parameter of sampling rate on data. Thus, in order to make domain-relevant error predictions, we must analyze the effect of data sampling with respect to physical error propagation. The application queries, shown in Table 1, serve as model and motivating examples in this study. We will discuss the two applications independently.

#### Table 1: Experimental Geospatial Queries

<table>
<thead>
<tr>
<th>DEM Query</th>
<th>Shoreline Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>“return land elevation change at (482593, 4628522) from 07/08/2000 to 07/08/2005”</td>
<td>“return shoreline extraction at (482593, 4628522) on 07/08/2004 at 06:18”</td>
</tr>
</tbody>
</table>

#### 3.1 DEM Elevation Difference Query

This query involves a simple algorithm which extracts the elevation between two digital elevation model (DEM) files \( DEM_1 \) and \( DEM_2 \). A DEM is essentially an \( m \times n \) grid with elevation, e.g., Z-values, at each point. Suppose that \( DEM_2 \) is sampled to reduce time cost, which effectively widens the gap between each pair of points. This variation between resolutions, as shown in the left-hand side of Figure 3, presents difficulties to the otherwise trivial computation of elevation difference, \( Z_{DIFF} = Z_{DEM_1} - Z_{DEM_2} \). In order to compensate for the unknown Z-values of \( DEM_2 \), interpolation becomes necessary. Each DEM now has disparate grid spacing which results in different errors along the horizontal and vertical directions.

It is worth noting that even without sampling, interpolation may be necessary to normalize the grid sizes between two heterogeneous DEMs (perhaps, measured by two data sources with different resolutions).

![Figure 3: The Overlay of \( DEM_1 \) and \( DEM_2 \)](image)

The Kriging method [7] is implemented for interpolation to calculate the elevation information corresponding to \( DEM_1 \) from \( DEM_2 \). Krige is a set of geostatistical techniques to interpolate the value \( Z_{inter} \) at an unobserved location corresponding to the grid location on \( DEM_1 \) from observations of nearby points on \( DEM_2 \). This method computes the optimal linear unbiased estimator of \( Z_{inter} \) based on a stochastic model of the spatial dependence quantified either by the variogram \( \gamma(x,y) \) or by the expectation \( \mu(x) = E[Z(x)] \) and the covariance function \( c(x,y) \) of the random field.

Depending on the stochastic properties of the random field different types of Kriging apply, but in our case, the Simple Kriging method applies,

\[
\hat{Z}(r_0) = \sum_{i=1}^{N} \lambda_i Z(r_i)
\]

which assumes the expectation of the random field to be known and relies on a covariance function. It is the weighted sum of the data set and are the weights which depend on the semivariogram, e.g., the distance to the prediction location, and spatial relationship among the measured location around the prediction value.

\[
\begin{bmatrix}
\gamma_{1,1} & \ldots & \gamma_{1,N} \\
\vdots & \ddots & \vdots \\
\gamma_{N,1} & \ldots & \gamma_{N,N}
\end{bmatrix}
\begin{bmatrix}
\lambda_1 \\
\vdots \\
\lambda_N
\end{bmatrix} =
\begin{bmatrix}
\gamma_{1,0} \\
\vdots \\
\gamma_{N,0}
\end{bmatrix}
\]
Finally, the error at each point for the difference calculation be given by:

\[ \sigma^2_{Z_{DEM_4}} = \sigma^2_{Z_{DEM_1}} - \sum_{i=1}^{N} \lambda_i \gamma_{i,0} \]

Finally, the error at each point for the difference calculation is summarized as

\[ \sigma_{Z_{DIFF}}^2 = \sigma^2_{Z_{DEM_1}} + \sigma^2_{Z_{DEM_2}} \]

### 3.2 Shoreline Extraction Query

The shoreline query involves obtaining the water level and a coastal terrain model (CTM) file for the targeted area and time. The CTM, as illustrated in Figure 4, is obtained by mosaicking underwater landscape (bathymetry) and a DEM over land [23]. To put simply, CTMs are not unlike DEMs, with the difference being that the points can represent both land surface and bathymetry values. A shoreline is obtained by intersecting water level with the CTM. Here, the effects of CTM sampling are not as complex as the former query. The missing data points are interpolated as an average of the samples, and a natural degradation of the shoreline’s quality is expected.

![Figure 4: Shoreline Extraction Process](image)

As shown in the above figure, the shoreline position would be given by:

\[ \Delta x = \frac{\delta x}{(h_2 - h_1)} \times (h - h_1) \]

where \( \delta x = (x_2 - x_1) \) and \( x_1 \) and \( x_2 \) are distances between grid points. \( h \) denotes the water level, and \( h_1 \) and \( h_2 \) represent grid point elevations. Then the error can be estimated as follows:

\[ \sigma_{\Delta x} = \sqrt{\left( \frac{\partial h_1}{\partial \Delta x} \right)^2 + \left( \frac{\partial h_2}{\partial \Delta x} \right)^2 + \left( \frac{\partial h}{\partial \Delta x} \right)^2} \]

which is reducible to

\[ \sigma_{\Delta x} = \sqrt{\frac{\Delta x^2 \sigma_{h_1}^2 (h_2-h_2)^2 + (h-h_1)^2} {(h_2-h_1)^2}} + \frac{|\Delta X \gamma_{i}^2 + (h-h_1)^2 \sigma_{h}^2|} {(h_2-h_1)^2} \]

In this study, the average water level for a year was calculated using the hourly water level observations available from NOAA [18]. The average slope along the shore was considered to compute the \( h_1 \) and \( h_2 \) values used in the above equation. Later, in Section 5, we discuss an evaluation of this error prediction method against an actual shoreline along the coast of Lake Erie near Sandusky, Ohio.

### 4. QOS WORKFLOW COMPOSITION

In this section we focus on problem formulation and implementation details of our approach.

#### 4.1 Problem Statement

In practice most workflows can be expressed as directed acyclic graphs where the vertices denote services and data elements and directed edges represent the flow of execution. Formally, workflows can also be recursively defined as follows. Given some arbitrary dataset, \( D \) and a set of services \( S \), a workflow \( w \) is

\[ \omega = \{ \epsilon, d \} \]

such that terminals \( \epsilon \) and \( d \in D \) denote a null workflow and a data instance respectively, and nonterminal \( (s, P_s) \in S \) where \( s \) denotes a service with parameter list \( P_s = (p_1, \ldots, p_k) \) and each \( p_i \) is itself a workflow. Then given a set of workflows \( W_q = \{ w_1, \ldots, w_n \} \) capable of answering some user query \( q \), our goal is to identify some subset \( R_q \subseteq W_q \) such that each workflow \( r \in R_q \) either meets or exceeds user constraints, namely, processing time and accuracy of results.

#### 4.2 Modeling Workflow Cost

We propose two cost functions, for aggregating workflow execution time and error propagation. The workflow’s time cost is estimated by

\[ T(w) = \begin{cases} \epsilon, & \text{if } w = \epsilon \\ t_{net}(d), & \text{if } w \in D \\ t_{net}(s, P_s) + \max_{p_i \in P_s} T(p_i), & \text{if } w \in S \end{cases} \]

If workflow \( w \) is a base data element, then \( w = d \), and the cost is trivially the data transmission time, \( t_{net} \). When \( w \) is a service, then \( w = (s, P_s) \), and its time can be summarized as the sum of the service’s execution time \( t_s \), network transmission time of its product, and, recursively, the maximum time taken by all of its parameters (assuming their execution can be carried out concurrently).

The error aggregation function, \( E(w) \), which represents the error estimation of a given workflow, is also in the familiar recursive sum form

\[ E(w) = \begin{cases} 0, & \text{if } w = \epsilon \\ \sigma(d), & \text{if } w \in D \\ \sigma(s, P_s) + \max_{p_i \in P_s} E(p_i), & \text{if } w \in S \end{cases} \]

Due to the heterogeneity of datasets and processes, it is expected that disparate workflows will yield results with fluctuating measures of accuracy. Again, at the base case
lies the expected error of a particular data set, \( \sigma(d) \). An error value can also be attributed to a service execution, \( \sigma(s, P) \). For instance, errors will be introduced if a sampling service is called to reduce data size or some interpolation/extrapolation service is used predict some value, e.g., the methods for deriving \( \sigma \) in the previous section handle this particular issue.

4.3 Workflow Enumeration and Pruning

As previously mentioned, our workflow enumeration (Algorithm 1) is based on Depth-First Search\(^1\). Starting from the target domain concept, we explore each dependent path in the given ontology until it leads to a sink, that is, base data node. All intermediate nodes between the target and data nodes are service nodes used to derive some intermediate concept.

Every concept (intermediate or target) can be realized by various datasets or services. (Line 6) obtains a set of “next” data or service nodes towards its derivation (null workflows are implicit in this algorithm). Each element in this set marks a potential workflow candidate either in the direction of available data or a service product. In the former case (Line 8) \( w \) is a base data workflow and immediately considered for inclusion. The latter case (Line 11) considers the service at-hand along with its parameters. Each service parameter is essentially a new subtarget concept in our ontology, and consequently, the algorithm is called recursively to solve for a set of its subworkflows.

For instance, consider a service workflow with two parameters of concepts \( a \) and \( b \): \( (s, (a, b)) \). Assuming that subtarget concepts \( a \) and \( b \) are derived using some set of subworkflows \( W_a = \{ w_1^a, w_2^a \} \) and \( W_b = \{ w_1^b \} \), then the workflows derivable from \( s \) includes \( W_s = \{ (s, (w_1^a, w_1^b)), (s, (w_2^a, w_1^b)) \} \). That is, if every service parameter can be substantiated with at least one subworkflow, then the full set of workflow compositions is established through a cross product of its derived parameters (Line 23). Each element from the cross product is then coupled with the service and considered for inclusion.

When a workflow becomes a candidate for inclusion, QoS-Merge (Algorithm 2) is called to provide the decision: prune, include as-is, or modify workflow accuracy then include. For simplicity, we consider a single error model, and hence, just one adjustment parameter in our algorithm, when in reality workflows may involve various error aspects for consideration such as the example given in Section 2.

QoS-Merge, which is simplified for clarity by excluding code for handling QoS precedence and \( \epsilon \) constraint buffer (as discussed towards the end of Section 2), inputs the following arguments: (1) \( W \), the set of current workflow candidates, (2) \( w \), the current workflow under consideration, (3) \( t' \) and (4) \( e' \) are the predicted time and error values of the workflow from the previous iteration (for detecting convergence), and (5) QoS is the QoS object from the original query. In summary this algorithm merges the given workflow candidate, \( w \), with the result set \( W \) if it meets time and/or error constraints.

---

\(^1\)Details on query parsing, the handling of immediate data values, and data identification can be found in [5]

---

**Algorithm 1** enumWF(target, QoS)

```plaintext
1: \( W \leftarrow \emptyset \)
2: /* static array for memoization */
3: global subWorkflows[...]
4: 5: /* B denotes the set of all data/service elements that can be used to derive target concept */
6: \( B \leftarrow \) derives(target)
7: for all \( \beta \in B \) do
8: if \( \beta \in D \) then
9: \( w \leftarrow \) data(\( \beta \))
10: \( W \leftarrow \) QoSMerge(\( W, w, \infty, \infty, \) QoS)
11: else
12: 13: /* \( P_\beta \) denotes the set of service’s params */
14: \( P_\beta \leftarrow \) getServiceParams(\( \beta \))
15: \( \Delta \leftarrow \emptyset \)
16: for all \( p \in P_\beta \) do
17: if exists(subWorkflows[\( p\).concept]) then
18: \( \Delta \leftarrow \Delta \cup \) subWorkflows[\( p\).concept]
19: 22: /* convergence of model estimations */
23: Params \( \leftarrow \) crossProduct(\( \Delta \))
24: for all \( pm \in \) Params do
25: \( w \leftarrow \) srvc(\( \beta, pm \))
26: \( W \leftarrow \) QoSMerge(\( W, w, \infty, \infty, \) QoS)
27: end for
28: end if
29: end for
30: subWorkflows[target] \( \leftarrow W \)
31: return \( W \)
```

**Algorithm 2** QoSMerge(\( W, w, t', e', \) QoS)

```plaintext
1: /* no time constraint */
2: if \( \) QoS.Time \( = \) \( \infty \) then
3: \( C_T \leftarrow \infty \)
4: else
5: \( C_T \leftarrow v \)
6: end if
7: /* no accuracy constraint */
8: if \( \) QoS.Err \( = \) \( \infty \) then
9: \( C_E \leftarrow \infty \)
10: else
11: \( C_E \leftarrow v \)
12: end if
13: /* constraints are met */
14: if \( T(w) \leq \) QoS.Time \( \land \) \( E(w) \leq \) QoS.Err then
15: /* insert \( w \) in QoS */
16: return \( (W \cup w) \)
17: end if
18: /* convergence of model estimations */
19: if \( |T(w) - t'| \leq C_T \land |E(w) - e'| \leq C_E \) then
20: /* prune \( w \) by excluding from \( W \) */
21: return \( W \)
22: else
23: \( \alpha \leftarrow \) getNextAdjustableParam(\( w \))
24: \( \gamma \leftarrow \) suggestParamValue(\( \alpha, w, \) QoS, \( C_E \))
25: \( w_{\text{adj}} \leftarrow \) w.setParam(\( \alpha, \gamma \))
26: return QoSMerge(\( W, w_{\text{adj}}, T(w), E(w), \) QoS)
27: end if
```
Initially, this algorithm assigns convergence thresholds $C_E$ and $C_F$ for error and time constraints respectively. These values are assigned to $\infty$ if its corresponding QoS is not given, and otherwise, $v$, some insignificant value. If the current workflow’s error and time estimations, $E(w)$ and $T(W)$ meet user preferences, the workflow is included into the result set. But if the algorithm detects that either of these constraints is not met, we ask the system to provide a suitable value for $\alpha$, the adjustment parameter of $w$, given the QoS.

Taken with the suggested parameter, the procedure is called recursively on the adjusted workflow, $w_{adj}$. After each iteration, $w$ is adjusted and if the constraints are met, is returned for inclusion. On the other hand, when the algorithm determines that the modifications to $w$ provide insignificant contributions to its effects on $T(w)$ and $E(w)$, i.e., the adjustment parameter converges without meeting QoS, then $w$ is subsequently pruned. This condition is shown on (Line 19) of Algorithm 2. The values of $t'$ and $c'$ of the initial QoSMerge call on (Lines 10 and 26) of Algorithm 1 are set to $\infty$ for dispelling the possibility of premature convergence.

Algorithm 3 shows suggestParamValue for handling error QoS, although time QoS is handled in much the same way. This algorithm assumes two cases: The trivial case is that a model is supplied for arriving at an appropriate value for $\alpha$, the adjustment parameter. Sometimes this model is simply inverse of either the time or error models which exists for solving $T(w)$ and $E(w)$. When this is not possible, such as in the case of our own experiments where complex calculations are involved towards arriving at $T(w)$ or $E(w)$, we must provide an efficient way for solving $\alpha$ in a forward fashion. This procedure essentially reduces to binary search where each iteration assigns a new value to $\alpha$ and subsequently checks for convergence. For this to work properly, of course, we must assume that $T(w)$ and $E(w)$ are monotonic. Depending on the granularity of each iteration, some overhead is expected with this approach, as will be shown in the experimental section.

If either $QoS.Time$ or $QoS.Err$ are not given by the user, their respective models are actually never invoked, and QoS-Merge becomes the trivial procedure of immediate inclusion of workflow candidate $w$. In Algorithm 2 this is equivalent to assigning the $QoS.*$ constraints to $\infty$.

5. EXPERIMENTAL RESULTS

Our system performance evaluation focuses on three goals: First, to evaluate the overhead of workflow enumeration and the impact of pruning. Secondly, we evaluate the efficiency and effectiveness of our parameter suggestion algorithm, and finally, we compare our shoreline’s error model with an actual degradation to validate our error model.

5.1 Overhead of Workflow Enumeration

Our initial goal is to show the efficiency of Algorithm 1. This core algorithm, called upon every given query, encompasses both auxiliary algorithms: QoSMerge — the decision to include a candidate and SuggestParamValue — the invocation of error and/or time models to obtain an adjustment value appropriate for meeting user preferences. Thus, an evaluation of this algorithm offers a holistic view of our system’s efficiency. A synthetic ontology, capable of allowing the system to enumerate thousands of workflows for a user query, was generated for this scalability experiment. The results, depicted in Figure 5, was repeated for an increasing number of workflow candidates (i.e., $|W| = 1000, 2000, \ldots$) on 4 configurations (solid lines) without pruning, i.e., the worst case scenario where all $|W|$ workflows meet any given constraint. These 4 settings correspond to user queries with (a) no QoS constraints, (b) only error constraints, (c) only time constraints, and (d) both constraints.

![Figure 5: Cost Model Overhead and Pruning Efficiency](image)

Expectedly, the enumeration algorithm runs in proportional time to the amount of models supported. It is also expected that the time cost model itself outweights the error model because it evaluates 3 distinct predictions, $t_x$, $t_{net}$, and $size_d$. To evaluate our algorithm’s efficiency, we altered our previous experimental setting to contain exactly 1 workflow within each candidate set that meets both time and error constraints. All other candidates are pruned in the early stages of workflow composition, which corresponds to the best case. For each setting of $|W| + 1$, the algorithm now prunes $|W|$ workflows (dashed line). The results show that the pruning algorithm is as efficient as, and later, begins to outperform the no-cost model since the amount of subworkflows to consider is minimized.
5.2 Effectiveness in Meeting User Preferences

Next, the geospatial queries that were given earlier in Table 1 will be used to demonstrate our system’s efforts for supporting user preferences. A summary of these queries were also given previously in Section 3.

Table 2: Suggested Value of Parameters (DEM Query)

<table>
<thead>
<tr>
<th>Ideal</th>
<th>Suggested</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acc %</td>
<td>Error (meters)</td>
</tr>
<tr>
<td>-------</td>
<td>----------------</td>
</tr>
<tr>
<td>10</td>
<td>8.052</td>
</tr>
<tr>
<td>20</td>
<td>7.946</td>
</tr>
<tr>
<td>30</td>
<td>7.911</td>
</tr>
<tr>
<td>40</td>
<td>7.893</td>
</tr>
<tr>
<td>50</td>
<td>7.868</td>
</tr>
<tr>
<td>60</td>
<td>7.859</td>
</tr>
<tr>
<td>70</td>
<td>7.852</td>
</tr>
<tr>
<td>80</td>
<td>7.847</td>
</tr>
<tr>
<td>90</td>
<td>7.8437</td>
</tr>
<tr>
<td>100</td>
<td>7.8402</td>
</tr>
</tbody>
</table>

We start with an evaluation of the parameter suggestion procedure. For these experiments, the sampling rate is the exposed workflow accuracy adjustment parameter, and error models for both queries have been defined as a function of sampling rate. First, we focus on DEM query. Table 2 shows the ideal and actual (system provided) error targets. On the left half of the table, the ideal accuracy % is the user provided accuracy constraint and the ideal error is the error value that is expected given this accuracy preference. The right half of the table shows the actual accuracy % and errors that the system provided through the manipulation on sampling rate. As can be seen, even though the error model appears to be extremely sensitive to ostensibly insignificant amounts of correction, our system’s suggestion of sampling rates does not allow a deviation of more than 1.246% on average, and 5.04% in the worst case.

DEM query was executed with user given accuracy preferences of 10%, 20%, ..., 100%. DEM files of sizes 125mb and 250mb were utilized to show the consistency of our system’s algorithm. As can be seen in Figure 6, the sampling rates along with the workflow’s corresponding execution times at each accuracy preference, increase as the user’s accuracy preference increases. The figure clearly shows the benefits from using sampling, as the execution time is reduced polynomially despite some loss in accuracy.

The above experiments were repeated for the shoreline query to obtain Table 3. Again, the results are consistent with the previous experiment, and moreover, our system offers slightly better parameter adjustments which results in tighter accuracies for this query. This can be explained again due to the fine-grained sensitivity of the error model for the previous query. We exhibit only a 0.07% average and 0.13% worst case accuracy deviation from the expected values. CTMs of sizes 125mb and 250mb were used to run the actual experiments. The results, depicted in Figure 7 again show the consistency of our algorithm and the effects of sampling on both workflow accuracy and execution time.

Table 3: Suggested Value of Parameters (Shoreline Query)

<table>
<thead>
<tr>
<th>Ideal</th>
<th>Suggested</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acc %</td>
<td>Error (meters)</td>
</tr>
<tr>
<td>-------</td>
<td>----------------</td>
</tr>
<tr>
<td>10</td>
<td>61.1441</td>
</tr>
<tr>
<td>20</td>
<td>30.7205</td>
</tr>
<tr>
<td>30</td>
<td>20.4803</td>
</tr>
<tr>
<td>40</td>
<td>15.3603</td>
</tr>
<tr>
<td>50</td>
<td>12.2882</td>
</tr>
<tr>
<td>60</td>
<td>10.2402</td>
</tr>
<tr>
<td>70</td>
<td>8.7773</td>
</tr>
<tr>
<td>80</td>
<td>7.6801</td>
</tr>
<tr>
<td>90</td>
<td>6.8268</td>
</tr>
<tr>
<td>100</td>
<td>6.1441</td>
</tr>
</tbody>
</table>

Figure 6: Workflow Accuracy and Corresponding Execution Times for DEM Query

Figure 7: Workflow Accuracy and Corresponding Execution Times for Shoreline Query
Next, we discuss the evaluation of the parameter suggestion overhead. Recall that the parameter suggestion algorithm has two cases: (1) trivially invoke a predefined model for solving for the adjustment parameter (in this case, the sampling rate), or (2) if this model is not available, it solves for the parameter through binary search on the sampling rate by employing $E(w)$ or $T(w)$ per sampling rate at each iteration. For both queries, error estimation, i.e., the term $\sigma$ in $E(w)$, involves a series of computations, and an inverse model cannot be easily derived. This forces the suggestParamValue algorithm to default to the latter case of binary search. The overhead (in msecs) to this approach is summarized in Figure 8.

![Figure 8: Overhead of Workflow Accuracy Adjustment](image)

Figure 8: Overhead of Workflow Accuracy Adjustment

Again, the sensitivity of the DEM query’s error model observes a slightly longer time-to-convergence. This overhead, however, contributes negligible time to the overall enumeration time, shown earlier in Figure 5. A quick study was also carried out to compare these overheads to the trivial case of model invocation. Surprisingly, the best case time for model invocation (model contains no calculations and simply returns a constant) cost 0.024 msecs, which is significantly more expensive. This cost can be explained through the heavyweight implementation of our model — that is, we utilize an equation parser to offer users an intuitive interface for inputting complex calculations. This flexibility, of course, is not without the cost of data structure management, which undoubtedly contributes to the overhead.

We believe that our experimental results suggest that the system maintains robustness against user defined cost, and although not shown due to space limitations, parameter adjustment to meeting time constraints exhibited similar results.

5.3 Shoreline Error Model Evaluation

Our final experiment evaluate only the shoreline error prediction model based on the availability of actual results for comparison. Recall that the time cost of shoreline extraction is dominated by retrieving and processing the CTM corresponding to the location. The sampling algorithm for DEMs and CTMs essentially skips $\lceil 1/r \rceil$ points per dimension, where $r$ is the sampling rate. In our sampling algorithm, the CTM is reduced by eliminating data points at a regular interval. Clearly, the shorelines obtained from different sampling patterns would contain errors. By taking exponentially smaller samples ($r = 100\%, 50\%, 25\%, 12.5\%, 6.25\%, 3.125\%$), we effectively double the amount of points skipped per configuration.

<table>
<thead>
<tr>
<th>Acc (%)</th>
<th>Error (meters)</th>
<th>Stddev</th>
</tr>
</thead>
<tbody>
<tr>
<td>100%</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>50%</td>
<td>1.36071</td>
<td>0.833924</td>
</tr>
<tr>
<td>25%</td>
<td>1.454593</td>
<td>1.050995</td>
</tr>
<tr>
<td>12.5%</td>
<td>2.651728</td>
<td>1.824699</td>
</tr>
<tr>
<td>6.25%</td>
<td>5.258375</td>
<td>4.06532</td>
</tr>
<tr>
<td>3.125%</td>
<td>15.03924</td>
<td>9.954839</td>
</tr>
</tbody>
</table>

Table 4: Actual Shoreline Errors

Given the sampled CTMs, we created a visualization of the resulting shoreline using ESRI ArcMap, depicted in Figure 9(a). Using the $r = 100\%$ setting as our baseline, it is visible that a slight deviation is associated with every downgraded sampling rate configuration. This becomes clearer in the zoomed region shown in Figure 9(b), which also makes visible the patterns of sampling and its deteriorating effects on the results. The actual errors shown in Table 4 are much less than predicted by our model (compare with Table 3). Admittedly, this suggests that our model may be excessively conservative, at least for this particular shoreline. While the initial consequence is that a smaller sampling rate could have been suggested by our system for speeding up workflows involving extremely large datasets, it does, however, ultimately demonstrate that the actual results are no worse than what the model predicts and that our framework is overall safe to use.

6. RELATED WORKS

In general, the class of dynamic workflow composition systems for supporting composite business and scientific processes has been studied extensively in a number of works [19, 27, 21, 14, 3, 24, 22]. These systems typically enable end-users to compose workflows from a high level perspective and automate workflow scheduling and execution. Fujii and Suda [13] developed a system to allow for automatic composition of workflows through the use of semantic information.

Workflow systems with QoS support have also been developed. Most works in this area concentrate on process/service scheduling in order to minimize total execution times. Eder et al. suggests heuristics for computing process deadlines and meeting global time constraints [11]. Other works, including Zeng et al’s grid workflow middleware [29], Pegasus [9, 15], Amadeus [4], Askalon [25], and more recently, stochastic modeling approaches [1, 26] exploit grid technologies, where datasets are inherently assumed heterogeneous and intelligent workflow scheduling on resource availability becomes a greater issue in meeting time constraints.

The notion and merits of utilizing service-oriented workflows, or so-called service chains, within the scope of geoinformatics were originally highlighted in [2]. However, to
truly realize the autonomous construction of workflows relies heavily on well-defined and standardized domain specific information. Consequently, studies on the use of geospatial ontologies for automated workflow composition have been carried out. The work of Lemmens et al. [17] describes a framework for semi-automatic workflow composition. Yue et al. successfully demonstrated that automatic construction of geospatial workflows can be realized using their ontological structure [28, 10]. Hobona et al. [16] combines a well-established geospatial ontology, SWEET [20], with an adopted notion of semantic similarity of the constructed workflows and the user’s query.

Our system differs from the above in the way that we assume multiple workflow candidates can be composed for any given user query. This set of candidates is pruned on the apriori principle from the given user preferences, making the workflow enumeration efficient. Furthermore, we focus on an accuracy-oriented task by allowing the user to specify application/domain specific time and error propagation models. Our system offers the online ability to adjust workflow accuracies in such a way that the modified workflow optimizes the QoS constraints.

7. CONCLUSION AND FUTURE WORK
In modern geospatial workflow composition systems, user preferences are becoming increasingly more desirable due to the possibility of having to move and analyze massive datasets. This paper reports an approach to enabling time and accuracy constraints in workflow composition through methods for modeling application-specific error and execution time prediction. We evaluated our system in many dimensions, and overall, our results show that the inclusion of such cost models contributes insignificantly to the overall execution time of our workflow composition algorithm, and in fact, can reduce its overall time through pruning unlikely candidates at an early stage. We also showed that our dynamic accuracy parameter adjustment is effective for suggesting relevant values for data reduction parameters, with a maximum difference of 5.04% between the ideal sampling error versus the system suggested errors between the two queried applications. We further showed that this parameter adjustment decision typically converged on the order of microseconds, again attributing negligible overhead. Finally, an evaluation of the shoreline error model was presented, and although we concede that our shoreline model offers conservative predictions since the sampled datasets far exceeded our expectations in retaining accuracy. It is, however, conclusive, that no real results were worse than the predicted, and our model is overall safe for use in practice, but perhaps, overly prudent.

As we seek to further our development of this system, we are aware of features that have not yet been investigated or implemented. One known area is workflow scheduling on distributed heterogeneous resources. The problem, which is inherently NP-Hard, has received much recent attention, and many heuristics have been developed to address this issue. Adding other features such as partial workflow caching, data/service migration, and fault tolerance, the scheduling problem is worsened. We propose to investigate the support for these aspects and heuristics on enabling an efficient and robust scheduler.

8. REFERENCES


