Task-Oriented Integrated Use of Biological Web Data Sources

Mustafa Kirac Ali Cakmak Gultekin Ozsoyoglu
Department of Electrical Engineering and Computer Science, Case Western Reserve University
{mustafa, cakmak, tekin}@case.edu

Abstract

Biological web data sources have now become essential information sources for researchers. However, their use is tedious, labor-intensive, repetitive, and possibly involve the integration of data from multiple web data sources. In this paper, as a first step towards the full integration of web data sources, we propose a framework that allows an integrated use of biological sources in a task-oriented manner. We define and experimentally evaluate a toolkit-based framework for semi-automatically constructing an integrated (software) system that automates and optimizes the execution of a biology-related computational task at hand. To test and refine the principles of the framework, we build and evaluate "Pathway-Infer" as a benchmark integrated system.

1. Introduction

In recent years, the number of biological web data sources on the web, and the quality and the quantity of information available to biologists have increased at a very fast rate [8, 9]. Such data sources are now essential to biologists. However, for computational tasks involving an answer to a biological question, biologists often extend large amounts of time and effort to (a) manually use the web interfaces of biological data sources, (b) extract and import data into their own environments, (c) relate various disconnected information found in the extracted data, (d) refine the search criteria, and (e) repeat the whole process from the beginning. The characterizing property of the computational tasks referred to here is that they all involve repeated access to biological web data sources, and integration/interpretation/analysis of the accessed data. We give an example.

Example 1. *(Inferring metabolic pathways).* Presently, there are a large number of organism-specific pathways on the web (e.g., Kegg [18] and aMaze [2]). Case Pathways Database (PathCase) [11] contains human and mouse metabolic pathways as well as pathways that do not directly belong to an organism (from [21]) (i.e., *generic* pathways). We would like to implement a software system, called *Pathway-Infer,* that allows users to infer organism-specific versions of a given pathway. In other words, the system allows users to execute tasks of the type

T: "*Find, using the biological data sources on the web, organism-specific versions of a generic pathway P*".

Pathway-Infer is an "integrated" system in that it integrates data obtained by repetitive querying/accessing of possibly multiple web data sources. The goal of integrated software systems is to facilitate a wet-lab verification process by providing the user with the data of interest through integration of relevant information, database comparisons, literature search, and other web-based searches. This serves in reducing the search space and provides a starting set of steps to be verified in the wet-lab.

In this paper, to illustrate the issues and as a running example of an integrated system, we build and experimentally evaluate the *Pathway-Infer* system. And, to simplify the presentation, we assume that *Pathway-Infer* integrates data from a *single* web data source, e.g., NCBI [18] or KEGG [22]. Our goal is to enable biology researchers to build an integrated software system which accesses multiple biological web data sources, integrates the retrieved data, and solves the task at hand. Our approach is toolkit-based and employs (i) human-assisted and task-oriented data and functionality modeling of biological web data sources--for *only* the information needed for a given computational task, and (ii) component-based and interactive integrated system construction. More specifically, in this research project, we build a framework and a toolkit for the following tasks.

1. (Data source) model (construction) toolkit for a data source allows the designer of an integrated system to model parts of a biological web data source in terms of entities, attributes, and relationships of the ER data model [13]. This tool allows the designer to extend the data model as and when needed. (We distinguish two types of individuals: the *designer* of the integrated system is a computational scientist who designs and builds semi-automatically the integrated system targeted to solve the given task. The designer needs to be knowledgeable about biology, and is *not* necessarily the end-user of the integrated system. The *end-user*, referred to as the *user*, is a biologist who uses the integrated system for the research problem at hand.)

Proceedings of the 18th International Conference on Scientific and Statistical Database Management (SSDBM'06)
0-7695-2590-3/06 $20.00 © 2006 IEEE

2. (Data source) functionality (construction) toolkit: Each web form/page/service of a data source represents a certain functionality for that data source. A toolkit is built to identify each data source function needed, and to define the associated input and output.

3. (DesignerDefinedFunction) DDF library: To generate an integrated system for a given task, various functionalities (e.g., output filtering, scoring, etc.) are defined and coded by the designer, and made available in the DDF library.

4. (Integrated task-based) system (construction) toolkit: This toolkit allows designers to put together the desired integrated system, component-by-component, using (i) the available data models of data sources, (ii) functionalities of data sources, and (iii) functionalities made available via the DDF library.

We make the following observations about the toolkit approach. First, the data models of web data sources should be extracted--with guidance from the designer and only for the task at hand. This involves discovering entity and relationship types and their instances, for a given data source. Second, the generated integrated system needs to be capable of optimizing its execution via a "task implementation plan". In this paper, we characterize various cost measures and discuss task implementation plan optimization.

Note that the integrated system needs to be "parameterized" in that it not only solves the task at hand, but also it is useful for a *class of tasks*. And, once an integrated system is built for a given task, extending it to similar tasks is likely to be easy—to the point that the user, not the designer, can revise it unaided. Moreover, the level of integrated systems is such that computationally-aware biologists as users are able to generate integrated systems for relatively simple tasks (and simple task changes). Finally, the integrated system needs to be enhanced with additional capabilities, namely, score management functionality, ontology modeling, and text management capabilities—to save space, these issues are not discussed in this paper.

The main contribution of this paper is to define and experimentally evaluate the first version of a toolkit-based framework approach for semi-automatically constructing a software system that automates a biology-related computational task at hand. To illustrate, test and refine the principles of the task-oriented biological web data source integration framework and to generate the first versions of the three toolkits and the DDF library, as a benchmark system, we discuss the generation of the *Pathway-Infer* system. Our long-range objective in this research is to develop a task-oriented biological web data source integration framework and a toolkit for a number of bioinformatics-related computational tasks, as well as for distribution to the research community.

The toolkit approach to solving biological problems is not new; see myGrid [36], BioSpice [3] and Seed [29]. The novelty of our approach can be summarized as follows. (a) It is targeted for task-oriented web data source integration in such a way that it can answer the immediate needs of biologists. (b) Once the toolkits are in place, the integrated system building effort will be relatively easy and automated. (c) The framework is firmly grounded in data modeling, providing an information-rich framework, allowing further advanced extensions to the toolkits such as adding data mining functions. Note that biological data sources on the web are autonomous--maintained and managed independently and with little coordination with each other. For flexibility and usability, we also develop a framework that does not rely on a cooperation from (the administrators of) data sources.

Section 2 elaborates on execution steps of the integrated system, *pathway-infer,* and presents data model, functionality model and task implementation plan representations. In section 3, we discuss designer-guided data model extraction from web data sources. Section 4 describes alternative task implementation plan discovery and optimization issues and, finally, in section 5 we present experimental results to evaluate the cost and optimization models described in section 4.

2. Inferring Metabolic Pathways

Task T of Example 1 can be implemented roughly as follows: given a catalyzing enzyme of a process p in a generic pathway P (of PathCase), the integrated system checks whether a homologous enzyme for organism o is known to exist in biological web data sources. If it does then the inference is that p also exists for organism o. Additionally, assuming that the process p for organism o has all the compounds participating in the generic process p (which is commonly the case), and repeating this procedure for each process in pathway P, the system infers the version of P specific to organism o. Next we list the implementation steps of task T using the existing functions of data sources.

Steps of Task T:

1. *Query Entrez:Gene, and locate all (Entrez:Gene) gene id's using either the enzyme name and/or its EC#.* For this step, there are two alternative ways: (a) query (using a web form) the NCBI E-utilities http site [22], or (b) query NCBI ESearch web services [23].Let the resulting gene set (with Entrez:Gene gene ids) be G_e. This step has several possible outcomes with potential problems: (i) generic enzyme names result in too many results, (ii) typos or incomplete enzyme names lead to no genes/proteins, (iii) no genes are returned for an enzyme. To solve these problems, step 1 is *interactive,* and includes a *feedback loop and filtering*. That is, the results are verified interactively by the user before the system moves on to the next step.

2. *Locate the gene product (protein) ids (gi) for each gene in G_e*. For this, there are two alternative ways: (a) download *gene2refseq* data file (available at [25]) that lists all protein gis for each gene id. This step is to be implemented by the designer, possibly as a designer-defined function, and stored into the DDF library. (b) Query using the NCBI ELink web services [23] to locate gene product ids.

3. *Locate the protein (amino acid) sequence for each protein gi* by querying NCBI EFetch web services [23].

4. *Get homolog gene products by running BlastP* (on a local machine) using the protein sequence (and properly-chosen parameters, such as "E-value") to obtain similar protein sequences. Call this set *HomGP*. In this step, the user needs to experiment with the proper input parameters, e.g., E-value of the blast search. Also, this step is likely to be time-consuming due to multiple blast searches.

5. *Get homolog protein genbank identity number gi for each gene product in HomGP*. Let the resulting set be *HomGPid*. This step is pre-implemented by the designer, and stored into the DDF library.

6. Obtain gene id for each gi in *HomGPid*. There are two alternatives: (a) Download and access the file *gene2refseq* (as in step 2). (b) Use NCBI ELink web services function (as in step 2). Let the resulting gene id set be *HomGid*.

7. Get detailed information for each gene (gene id in *HomGid*) and gene product (gi in *HomGPid*) from the *gene2refseq* file and the *gene_info* file [25].

Currently, the steps itemized above can only be done by the user directly accessing each data source, manually analyzing each output (likely to contain thousands of entries), manually extracting the data for the next query, and posing the next query to the next data source. Considering that each input to the next web data source needs to be entered/processed one-by-one through a web form interface, obtaining the results is labor-intensive, and takes a long time.

Figures below present (a) (parts of) the ER data model (i.e., entities and relationships) used to implement task T (Figure 1), and functions provided by the data source (i.e., NCBI for our running task) and DDF library (Figure 2.a-c), and (b) the task implementation plan, i.e., functionality implemented at each step, possible feedback loops and filtering (indicating returns back to earlier steps), and the needed interaction points (Figure 3). We make the following observations. First, from Figure 1, the data model needed to implement task T involves few entities and relationships, and is not complicated. Second, the functional data model of Figure 1 and the task implementation plan of Figure 3 can be defined semi-automatically with assistance from the designer relatively easily (i.e., *designer-assisted data and functionality modeling*). Third, once the data model and functionality are defined, with careful system-building techniques, they can be extended to other tasks by adding new data model entities and relationships, new data source functions, or new designer-defined functions (i.e., *extensible data and functionality modeling*). Fourth, the integrated system gets feedback from the user and filters its output at any step (i.e., *interactive system with feedback loops and filtering*). Fifth, in step 1, in addition to Entrez:Gene, another data source about gene ids may be used to enrich the output of step 1 (i.e., *entity enhancements*). Task T demonstrates the advantages of task-based integrated systems employing autonomous data sources with a specific research task in mind. We have illustrated with an example that developing a component-by-component solution (one component per task implementation step) for the integration of biological web data sources is a promising research direction, considering the benefits and the usefulness of the resulting integrated system.

A web data source provides a two-folded view, i.e., data stored in the data source, and a set of functionalities made available through web forms. And, in order to build a task-oriented integrated system based on the available functionality, a task implementation plan needs to be constructed. In the following section, we discuss the representation and formalization of such structures for the *Pathway-Infer* system.

2.1. Data Model

To simplify presentation, *Pathway-infer* is constructed using the data and the functionality provided by only one data source, namely NCBI. Figure 1 illustrates the data model of NCBI, involving only those data components which are required for Task T. In this model, entities, attributes, and relationship between the entities are represented in ER notation [13]. Each entity name is preceded by the data source name, and concatenated with type information which is required for modeling the functionality of web data sources. As discussed in Section 3, the data model can be extracted in a designer-assisted manner.

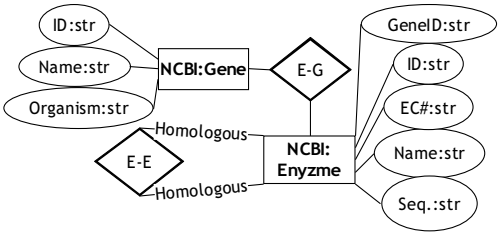

Figure 1. NCBI Data model for Task T

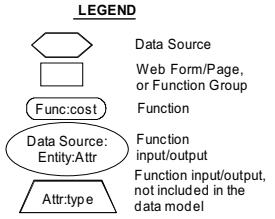

Figure 2. (a) Legend forFunctional Models

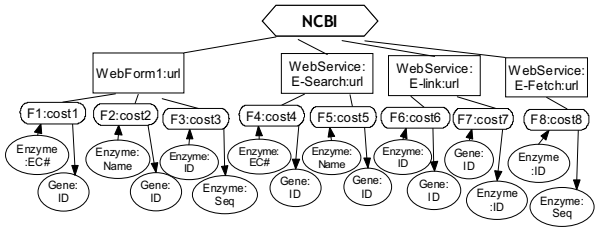

Figure 2.(b) NBCI Functional Model for Task T

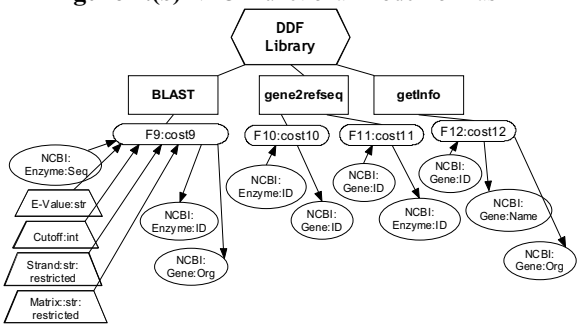

Figure 2.(c) DDF Library Functions Required for Task T

2.2. Functional Model

For functionality modeling of a given data source, each related web form/page/service of a data source is represented as a set of functions with the guidance of the designer. We model the functionality of a web data source via a hierarchical structure where each data source contains the web forms/services/pages, and finally, the set of functions are provided by each web forms/pages/services. Figure 2.(b) illustrates the functional model for NCBI. Note that, in figure 2(b) the functions listed under each origin (e.g., BLAST, gene2refseq) represent different querying capabilities of the data source and have different input/output specifications. The same functionality can be found at different web origins. In this case similar functions with same input and output specifications can be used as alternatives (See example 2).

Example 2. In Figure 2.(b), F1, F2 and F3 represent different functionalities of the WebForm1 and function F4 of E-search web service can be used as an alternative to F1.

The designer creates a function by specifying its *name*, *url* of its origin, and *input/output* parameters which are mapped to an attribute or a set of attributes of the entities represented in the data model of a given web source. Occasionally, some functions may require input parameters which are not included within the data model for a specific task (e.g., E-Value for blast function in DDF library, function F_9). In order to specify the nature of input and output of the functions in terms of the components provided in the model, we denote them as *[entity name]:[attribute name]*. Besides, each function has an associated cost value as discussed in section 4. We also represent the DDF library like a web data source without a data model specific to itself (i.e., it uses the available data models of web sources to type its function input/output parameters). Sample DDF library functions required for Task T are demonstrated in Figure 2.(c). The entire model is the same as that of a web source except the fact that, instead of web forms or services, DDF library uses, designer defined function containers which organizes similar functions into the same named group. Note that the presented function set is not complete in that there are many other functions included in the library, and used in the context of *pathway-infer*, but have not been shown here due to space limitations.

2.3. Task Implementation Plan Representation

Once the designer has the functional and data models for a given task, the implementation plan for the task can be constructed through designer guidance. The task implementation plan defines the sequence of functions to be executed, and the order of execution so as to complete the task T. Therefore, each *step* of the implementation plan corresponds to a function or, a set of functions where each function provides an alternative (e.g., step 2 in Figure 3), or enhances the output for that *step* (e.g., step 1 in Figure 3). Multi-function steps require branch connectors. The designer can put *AND* connector to mark that those branches are sub-steps of the current step or, *XOR* to mean that it is sufficient to execute only one of them. The task implementation plan for *pathway-infer is* demonstrated in Figure 3.

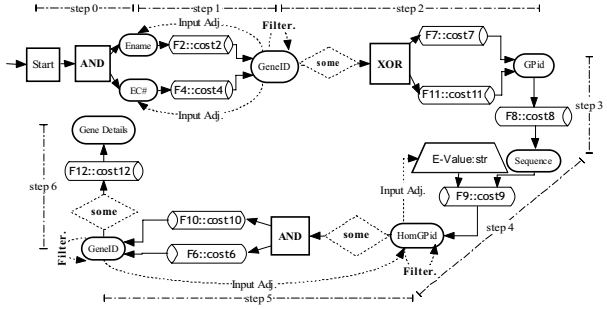

Figure 3. Task implementation plan for task T

For most of the biological web data sources, it is usually the case that a search process through a web form results in multiple candidate results, mostly because (a) keyword search is preferred, and secondly, (b) on a web form, filling out all the input parameters is not usually required. Thus, providing some filtering mechanisms allows for a flexible and powerful integrated system and extends the overall query capabilities of data sources. Such an extension can happen in two different contexts: *(a)* manual selection of the subset of the results obtained at the end of a step, and *(b)* filtering through designer-defined predicate enforcement (*e.g.*, selecting EC(enzyme commission) numbers that fall into a certain range). Default behavior is that, after the completion of a *step*, the integrated system executes the next step by providing each object/value in the result set as a parameter to the corresponding function of the following step. If the designer wants to enable users to analyze and possibly

filter the results manually at the end of a step, the keyword, *"some",* is used to mark filtering points. In Figure 3, for instance, step 2 is marked with "*some*" referring to the fact that the result set of*" gene id"* values will be filtered before they become inputs to the functions f_6 or f_{11} (as input parameters). The interactivity between the integrated system and the user is represented with dotted lines in Figure 3.

3. Data Model Extraction

Before the construction of the task implementation plan, data models of web data sources should be extracted with guidance from the designer. In this section, we describe the key steps of web data source model extraction from web forms (due to limited space). A similar model extraction design steps can be also described for web services using the WSDL files [38].

3.1. Entity Type and Instance Discovery

A web form query result page contains instances of one or more entity types. Firstly, the designer locates the web forms that would possibly be involved in the task implementation. Next, sample query parameters are submitted to the corresponding web query form. These parameters are provided by the designer in a sample query by filling out the corresponding web query form elements. We give an example.

Example 3. In our running example, one of the alternatives for the first step involves finding genes by querying NBCI with enzyme name as a parameter. Figure 4 illustrates NCBI Entrez Gene web form. Our sample query, in this case, is "*find all genes associated with name 'kinase'",* where the query parameter is the value '*kinase*'. Usually, this query returns a set of intermediate results (Figure 5), and clicking to one of the objects among the intermediate results brings a final result page that describes the details of a particular gene instance and is marked as an *entity* page (Figure 6).

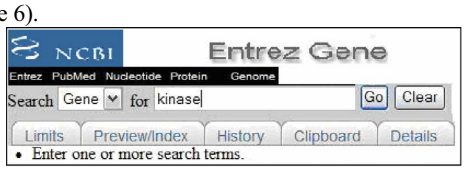

Figure 4. Sample query posed to NCBI Entrez Gene web form

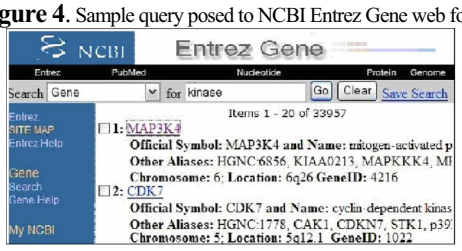

Figure 5. Intermediate results obtained for the sample query in Fig.4

The query scheme described in example 3 is valid for most of the biological data sources; however, there are also other data sources that do not conform to this paradigm (e.g., [17]). To simplify the presentation, in this section, our discussion does not take into account such data sources that provide different querying models.

Having located the entities as illustrated in example 3, next, we attempt to find out the attributes of the located entities. To this end, our goal is to automate the attribute-value extraction process. Our approach is to compare different instances of the same entity (*i.e.*, *different results of the same web query form*) and determine the repeating and varying parts of the output web pages. Then, repeating fields (e.g., gene name, organism) are presented to the designer as candidate attributes, and the designer selects the related attributes for the entity. Another alternative, but more naïve approach, is that the designer marks the required attributes manually, and the selected attributes are added to the data model. We give an example.

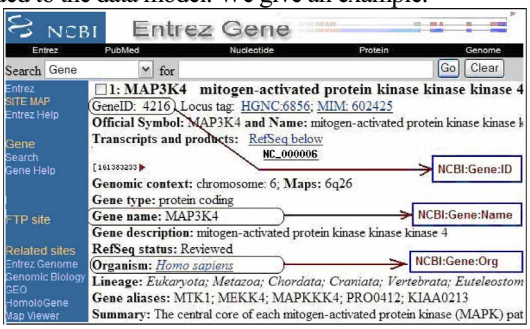

Figure 6. Gene details result page with marked attributes of Task T

Example 4. Figure 6 shows the attribute names of *gene* entity that are marked by the designer as the ones needed in task T. Although there are other attributes of the gene entity type, relying on the task-oriented motivation of our framework, the designer selects only those attributes that are interesting to him/her in the context of the *pathway-infer* system. The selected attributes (shown as framed in a rectangle in Figure 6) are also named by the designer.

3.2. Relationship Type and Instance Discovery

Another main component of a source data model is the relationships between the entities. We observe two types of indicators towards the existence of a relationship between the entities: *(a)* direct links between final query result pages (*e.g.,* a hyperlink from *pathway* result page to *reaction* result page), *(b)* un-hyperlinked common attributes, i.e., an entity attribute that can be used to query another entity through one of the available web forms. For the latter, to discover such implicit links, one approach is to locate the common attribute types belonging to the instances of different entity types based on comparisons of their names, and the percentage of value overlaps in a sample set of instances. To illustrate the utilization of direct links we give an example.

Example 5. Data model of NCBI for task T contains two *relationships* (Figure 1.a, relationships E-G and E-E). The one between enzyme and gene, i.e., E-G, can be inferred from the direct links on enzyme (*i.e., protein*) and gene result pages. Figure 7 shows the direct links between gene 'amn' and its gene product, 'amp nucleoisidase', in NCBI result pages.

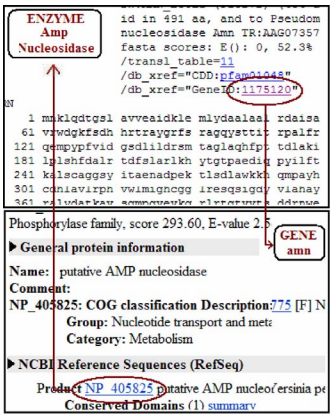

Figure 7. Enzyme-Gene association detection in NCBI

4. Task Implementation Plan Optimization

A generated task implementation plan may have lower-cost alternatives which can be constructed using other functions in the function repository (either the functionality toolkit or the DDF library). In this section, we define several cost notions, and discuss alternative plan discovery process and optimization issues.

4.1. Cost Evaluation and Measures

In order to discuss task implementation plan optimization issues, one needs to define cost measures to compare different alternative plans. We start by defining a set of cost measures for the functions which are the building blocks of a task implementation plan. For (query) optimization, traditional relational database management systems provide known properties and statistical information related to the various aspects of the stored data, which are used in (query) optimization. In comparison, for task plan optimization, such rich summaries (e.g., cardinalities, histograms, etc.) are not available for the integrated system we study, where queries, in the forms of web forms/services, are submitted to multiple heterogeneous and autonomous web data sources with different characteristics and capabilities. Given these challenges, two different approaches may be employed so as to model function costs. One is time-based cost measures, and the other is result-cardinality-based cost measures, i.e., average number of objects returned by a function. In this section, we only consider time-based cost measures. Next, we define the specific measures of our function cost model.

- *Average Response Time (RT):* The average time elapsed from the submission of a query (as a function) till the return of the first output object.
- *Average Connection Time (CT):* Average time to connect to the data source and to submit the query on a web form.
- *Average Result Download Time (DT):* After a query is posed to a web data source, results may be returned in a single XML file, a single web page, or multiple web pages. Distribution of results to multiple pages brings additional overhead to the overall execution time of a function. In order to capture this behavior, we define *query result download time* as the elapsed time from the retrieval of the first result object till the last fetched object.
- *Average Output Processing Time(OT):* Some web data sources might provide their output in such a format that may need pre-processing to get the desired function output as specified in the function definition (e.g., extracting a set of EC numbers from a returned flat file).

Cost of a function is formulated as a weighted sum of the values for each measure so that the designer can adjust the *weight* parameters in the cost function in a way that best fits to his/her needs. The general structure of the cost function can thus be characterized as follows: $fcost_i = w_{CT} * CT + w_{RT} * RT + w_{DT} * DT + w_{OT} * OT$, where w_i is the corresponding weight assigned by the designer to each measure. Next, the cost of step i in the task implementation plan is computed based on the functions f_j involved in that step and the connector between the functions (e.g. '*XOR*').

$$scost_i = \begin{cases} \max_{1 \le k \le j}(fcost_k) & if \quad j>1, step_{conn} = AND \\ \min_{1 \le k \le j}(fcost_k) & if \quad j>1, step_{conn} = XOR \\ fcost_i & if\ j=1 \end{cases}$$

In the step cost formulation, on the left, j stands for the number of functions involved in step i, and $step_{conn}$ indicates the connector between the functions of a particular step. Note that, when the functions of a step are connected by '*AND*', the cost of the step $scost_i$ is that of the function $fcost_k$ which has the maximum cost among k functions in the step, $i \le k \le j$, assuming that the functions can be executed in parallel within different threads. For the case of connector being *XOR*, a similar reasoning is employed.

Finally, the cost of a path consisting of a sequence of steps is computed by simply summing up the costs of individual functions each of which is multiplied by the expected number of tuples (objects) obtained from the previous step and are to become input to the function at hand. We obtain the cost of a task implementation plan *pcost* recursively as: $pcost=P(1,n)=scost_i+RC_i * P((i+1),n)$ where n is the number steps in the plan, $P(i,j)$ is the cost of the subplan starting from step i and ending at step j (inclusive), and RC_i is the result cardinality of step i. We give an example.

Example 6. Concerning the cost of the task implementation plan (see Figure 3) for our running example, *pathway-infer,* for a single execution with a single enzyme name/EC number, we have the following running times in seconds: $fcost_{F2}$=0.542, $fcost_{F4}$=0.446, $fcost_{F6}$=2.122, $fcost_{F10}$=0.513, $fcost_{F11}$=1.168, $fcost_{F7}$=0.412. Finally, using the observed RC values at each step, the resulting step cost assignments are as follows: $scost_1$ = max($fcost_{F2}$, $fcost_{F4}$) = 0.542, $scost_2$ = min($fcost_{F7}$, $fcost_{F11}$) = 0.412, $scost_3$ = $fcost_{F8}$ =1.863, $scost_4$ = $fcost_{F9}$=45, $scost_5$ = max($fcost_{F6}$, $fcost_{F10}$) =2.122, $scost_6$ = $fcost_{F12}$=1.118. Thus, the cost of the plan becomes:

$$pcost_{T1} = fcost_{F_2} + RC_1 * \left(fcost_{F_7} + RC_2 * \left(fcost_{F_8} + RC_3 * \left(fcost_{F_9} + RC_4 * \left(fcost_{F_6} + RC_5 * fcost_{F_{12}} \right) \right) \right) \right)$$

= 0.542+20*(1.863+2*(45+1*(2.122+15*(1.118)))) = 2,594 seconds = 43.2 minutes.

We associate each function with two variables: its estimated cost and estimated cardinality. The designer needs to provide sample inputs for each function to compute initial values for the estimated cost and cardinality variables.

4.2. Discovering Alternative Task Plans

After the designer constructs the first implementation plan for a specific task, it may be possible to build better alternative task implementation plans. Two different sources lead to alternative implementation plans: (*a) f*unction(s) marked as interchangeable by the designer, (*b) f*unction(s) that are inferred to be interchangeable from the available function matchings. Ultimately, the system presents the designer with a set of alternatives that make the overall plan less costly.

4.2.1. Designer-defined Equivalences. The designer may provide the system with a set of *semantic equivalence rules*, X≅Y, which define functions (*or function compositions*) X and Y as *semantically related*. Semantic equivalence rules define function *partitions* where each member of a partition can be replaced by any member of the same partition in a task implementation plan. If two functions/function compositions are defined to be semantically related, then they must have the same signature, i.e., the same type of input/output parameters. We give an example.

Example 7. Assume that the following semantic equivalence rule set $\Re$ is given by the designer: $\Re$ = {*F1=F2, F3=F4, F5=F2•F3*} where • denotes composition, i.e., pipelined functions executed one after another, e.g., *F2•F3* where *F3* is executed on the output of *F2*. Assume that a part of the overall task implementation plan is performed by a subplan *sp = F2•F3.* Based on the given rules, the system replaces some of the functions individually (*e.g.,* replacing *F2* with *F1*), or as a group (*e.g.,* replacing *F2* and *F3* with *F5*) with the other functions which are defined to be included in the same partition by the given rules. As such, alternatives are constructed, and lower-cost plans are added to the set of candidate task implementation plans. Note that, given some equivalence rules, finding the top-k optimal alternatives may be exponentially expensive in the number of functions involved in the rules. However, we assume that, in a typical integrated system, most of the time, the number of functions involved in equivalence rules is small.

A semantic equivalence rule can be defined by a simple grammar *G = (Vocabulary, Terminals, ProductionRules, StartState)* as follows. *Vocabulary* = {*S*, *OP*, *expr*}, *Terminals* = {*func*}, *ProductionRules* = {*S→ expr OP expr, expr→func • expr | func, OP →*⊃ | ⊇ | ⊂ | ⊆ | ≅ | =}, *StartState* = *S* where all operators in *OP* define a semantic equivalence between functions. '≅' is the most general semantic equivalence operator, while other operators in *OP* are specific versions of the general semantic equivalence operator in that they convey additional information on the quality of the relationship between the outputs of two functions (*compositions*) involved in a semantic equivalence rule. For instance, X⊂Y states that X and Y are semantically related, and given the same input, the output of X is always a subset of the output of Y. Moreover, if X *OP* Y is given, then X≅Y always holds where *OP* is any semantic equivalence operator.

4.2.2. Inferred Equivalences. Based on the existing equivalence rules, the integration system attempts to infer new rules. To achieve this, we introduce a set of axioms that can be applied repeatedly to infer all the rules implied by the available ones. Axioms illustrated in Figure 8 are always applied on the members of the same function partition except for the augmentation. Moreover, the axioms excluding *augmentation* never create new function partitions whereas they add new members to the partition that they apply. On the other hand, *augmentation* induces new function partitions by creating compositions among the members of different partitions. By generating a new partition, we attempt to discover additional semantic equivalences between newly composed functions.

- *Reflexivity* is trivial, e.g. if X =Y then Y = X.
- *Augmentation:* If X *OP* Y where *OP*∈ {⊃ | ⊇ | ⊂ | ⊆ | ≅ | =}, then for arbitrary functions Z and Q that can be composed with X and Y (i.e., compositions X•Z, Y•Z, Q•X, Q•Y are valid), X•Z ≅ Y•Z, and, Q•X ≅ Q•Y. In the special case of *OP* being '=', augmentation propagates complete equality (=) as X•Z = Y•Z and Q•X = Q•Y.

Commutativity: If X ⊃ Y, then Y ⊂ X, *or vice versa* and, if X ⊇ Y, then Y ⊆ X, *or vice versa*

- *Transitivity:* If XOP_1Y and YOP_2Z, then XOP_3Z where

$$OP_3 = \begin{cases} OP_1 & if\ OP_2\ is\ '='\\ OP_2 & if\ OP_1\ is\ '='\\ OP_1 & if\ OP_1 = OP_2\\ \subset & if\ OP_1\ is\ '\subset'\ and\ OP_2\ is\ '\subseteq',\ or\ \text{vice versa}\\ \supset & if\ OP_1\ is\ '\supset'\ and\ OP_2\ is\ '\supseteq',\ or\ \text{vice versa}\\ \cong & otherwise \end{cases}$$

Figure 8. Axioms used for inferring new semantic equivalence rules

Example 8. Assume we have the same set of rules given in the previous example, i.e., $\Re$ = {*F1=F2, F3=F4*, *F5=F2•F3*} and the subplan for which we generate alternatives, consists of single function *sp = F5.*

F1=F2 (given)
F1•F3 = F2•F3 (Augmentation)
F2•F3 = F5 (Commutative Property)
F1•F3 = F5 (Transitivity)

Similarly, we can derive additional rules, and obtain the extended rule set as $\Re$' = {*F1=F2, F3=F4*, *F5=F2•F3, F1•F3=F5,*

F2•F4=F5, F1•F4=F5}. Alternative plan set then becomes $\wp$= {*F2•F3, F1•F3, F2•F4, F1•F4*} each of which is compared to *sp* = *F5* in terms of the overall cost. Then, less costly alternatives are presented to the designer.

4.2.3. Generating Plans using Equivalence Rules. Using the designer-defined and inferred equivalence rules, we generate alternative task implementation plans. The goal here is that, based on the semantic equivalence information, the system attempts to find lower-cost alternatives for the task implementation plan constructed by the designer. For this purpose, after generating the possible plans, the system computes the estimated cost of each alternative, and presents the designer with a small set of "optimal" plans. Next, Figure 9 provides an algorithm sketch to illustrate how alternative plans are generated using the equivalence rules.

```
All_Alternatives = {original plan};
Repeat
Pick a plan P from All_Alternatives that is not processed before,
and mark it as processed;
for each equivalence rule X op Y do
    if X is a subsequence of the plan P, then
      {Replace X with Y;
       Add the resulting alternative plan into the set All_Alternatives;}
Until there is no plan found in All_Alternatives that is not
processed;
```

Figure 9. Alternative plan construction

Note that, the execution cost of this algorithm may become exponential in the number of equivalence rules in the worst case. However, we assume that, in practice, the number of equivalence rules provided to the system will not be very high.

Once the alternatives are generated, the designer may either keep the original task implementation plan, or choose among the lower cost alternatives. After the designer completes the construction of the task implementation plan, and selects an optimization method, lower cost optimal plans are generated accordingly. We define three different optimization criteria: minimum cost, maximum cardinality, and minimum cost per output (See Section 5).

There are two scenarios to *maintain an optimal task implementation plan*. One approach is, if desired, the integrated system can update the estimated cost and cardinality values of functions while the user is using and running the system. According to new estimated cost and cardinality values, if the current task plan is no longer an optimal one, it is dynamically replaced with the optimal version. Another approach is, the designer may want the task implementation plan to remain static and change occasionally. In this case, the estimated cost and cardinality values of functions need to be recomputed periodically by the designer. One can then use the following formulas to update the estimated cost and cardinality values.

$fcost^{j} = \alpha.\ fcost^{j-1} + (1-\alpha) fcost^{new}$ and $fcard^{j} = \alpha.\ fcard^{j-1} + (1-\alpha) fcard^{new}$, where $fcost^{j}$ and $fcard^{j}$ are the estimated cost and cardinality values computed at time j, and α is the coefficient for tuning the weight of older values in the final value.

5. Experiments & Results

We have implemented Task T functions using a single data source, NCBI. For each step in Task T, we developed multiple alternatives by employing different data access facilities provided by the data source. NCBI serves its data via web forms and web services. In addition to this, we also used NCBI's "Entrez Programming Tools" in which one can run database access commands using HTTP GET methods and the result is obtained in XML format or one of several other formats.

We needed an input set for each function to analyze the number of outputs and execution times. We have two particular functions in our function library, each of which takes a keyword as input, and returns the corresponding protein/gene ids of protein/gene entities containing the keyword as a substring. We probed these two functions with all possible strings of length two (i.e., aa, ab,…, zy, zz) and collected two large sets of protein and gene ids. Next, we created a smaller set by randomly picking 1000 ids from each of the large sets and used the smaller set as input to our other functions to collect further data instances consisting of various attributes such as protein, gene and organism names, EC numbers and so on.

In our experiments, we set the weights in the cost function (See Section 4.1) to 1. Since all of our functions use remote data sources and server loads vary over time, we computed average execution times of the functions at different hours and days. Sampling at different time points did not cause function execution times to change noticeably with respect to each other, since network delay affects all functions equally.

Finally, we pipelined necessary functions and prepared a task implementation plan for Task T. Then, we manually generated rules defining the equivalences between the functions. Without loss of generality, we modified the task implementation plan in Figure 3 so that we had a single function at each step by picking only one of the functions merged with AND and XOR operations. BLASTP is the most expensive function in the task plan. A single BLASTP function takes between 30-60 seconds depending on the query sequence. Hence, for experimental purposes, we replaced BLASTP computation with NCBI BLink operation which finds homolog proteins by using pre-computed sequence alignments and is 50 times faster than BLASTP operation. The task plan with Blink is still semantically the same task plan employing BLASTP. In practice, it is easy to replace BLink functions with BLASTP functions, later.

In our experiments, we filtered the results of the functions as follows. Functions taking keywords as input are calibrated to return only top 20 results (or less) which are the best substring matches with the input. For BLink functions, we retrieved only the top 15 results with highest similarity scores when more than 15 results are obtained.

We estimate the execution time and the number of outputs of a task implementation plan by using estimated execution times and estimated cardinalities of functions.

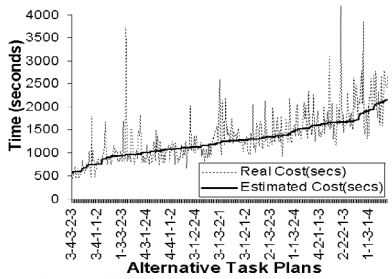

Figure 10.(a): Estimated vs. Real path execution time for each path alternative

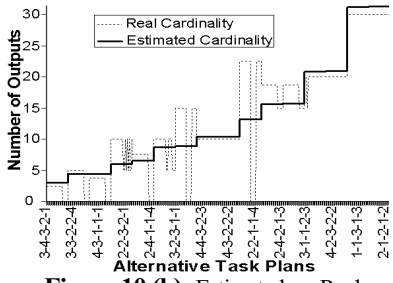

Figure 10.(b): Estimated vs. Real cardinality of each path alternative

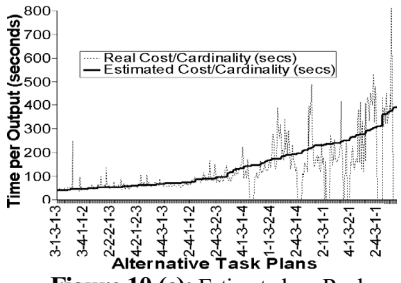

Figure 10.(c): Estimated vs. Real execution time per output for each path

Alternative plan generation algorithm (see Section 4.2.3) detects several alternatives of the functions at each step of the original task implementation plan. The algorithm found four alternatives of $step_1$, four alternatives of $step_2$, three alternatives of $step_3$, two alternatives of $step_4$, four alternatives of $step_5$ and a single alternative for $step_6$. Thus the alternative plan generation algorithm produced 4x4x3x2x4x1=384 alternatives for the implementation plan of Task T. Due to the availability of many alternatives for each task implementation plan step, the number of alternative plans grows exponentially. However, given the estimated cost and cardinality values for each function, calculating the estimated cost and cardinality of alternative plans can be done in less than a second on an average computer (P4 2.4-1GB) since little computation is required for each alternative, and the plans are not too long. For experimental purposes, we also executed each of 384 task plans with a small input set (5 protein names) to observe how good our estimations are.

For Task T, we generated 384 different alternatives. Only the names of 15 alternative task plans appear in between every 25 plans along the x-axises of the graphs in the figures. To distinguish the alternative task plans, we enumerated the step alternatives in the result of the alternative plan generation algorithm. Numbers in the alternative plan names specify which step alternative is used for a particular step. For instance, 2-3-2-1-2 is used as the name of the plan that consists of 2^{nd} alternative of 1st step, 3^{rd} alternative of 2nd step, 2^{nd} alternative of 3rd step, 1^{st} alternative of 4th step and 2^{nd} alternative of 5th step of the original task plan. For the 6th step, there is only one alternative, so we do not consider the 6th step in plan names.

Figure 10.(a) shows the estimated execution times of different task plan alternatives. We sorted the task plan alternatives by their estimated costs to observe the cost increase by alternatives. The estimated execution time increases exponentially since it depends on the number of outputs for each step, and is proportional to the multiplication of the number of outputs from all steps. Task plan with the minimum estimated execution time is the time-optimal-plan.

Figure 10.(b) shows the estimated cardinalities of each alternative task plan. In figure 10.(b), we find alternative task plans producing the maximum number of results, and then, from among these plans, we can pick the one with the smallest estimated execution time as the cardinality-optimal-plan. We sort the task plan alternatives by their estimated cardinalities to show the correlation between real and estimated cardinalities.

Figure 10.(c) shows the execution time of each alternative task implementation plan divided by the estimated number of outputs of the plan. In Figure 10.(c), we can trace how the time needed to produce a single output varies by changing the alternative plan. Highest peak in this graph shows the time/cardinality-optimal-plan which produces high number of outputs in low estimated execution times. In Figure 10.(c), alternatives are sorted by the estimated cost per output. Also note that, alternative task plan orders in the x-axis of the graphs are different in all three figures 10.(a), 10.(b) and 10.(c). Therefore we find three optimal plan sets for different optimality criteria.

6. Related Work

Web data source integration in general is studied by many researchers and involves mostly the integration of commercial, single-primary-entity web data sources (e.g. book, car, real estate) [20, 16, 15, 14]. As for the integration of biological data [26], several integration systems have been built. Earlier prototypes such as TAMBIS [30], DiscoveryLink [31] and BioKleisli [32] focused on querying facilities of an integrated system rather than the integration task itself.

myGrid [36] is a recent large scale application integration middleware for bioinformatics services. In myGrid, services (i.e., applications) are designed with open grid services architecture to ensure interoperability and reusability of the services. In addition, myGrid employs DAML+OIL [5] to conceptually describe its services and facilitate their discovery. In our work, we define the design steps of smaller-scale focused data integration tasks and we prefer the reusability of the finalized task rather than the smaller task components. Thus we picked a simpler model, namely ER model [13], as the underlying data model of web data sources in order to reduce the development time.

Taverna [27], Kepler [34], BioPipe [35] and JOpera [4] are open-source data source integration systems using workflow semantics and are similar to our system in terms of piped execution of heterogeneous functions (i.e., actors in workflow semantics). Taverna system [27] incorporates

user input to locate the workflow component alternatives to combine the results from alternative workflow components. In our system, we employ equivalence rules to check the integrity of a workflow (i.e., task execution plan), and optimize the original workflow by generating alternative workflows. None of Taverna, Kepler, BioPipe and Jopera provides workflow optimization.

Query optimization is one of the most extensively researched areas of database domain [19]. Workflow optimization (i.e., task execution plan optimization) that takes place in our work is similar to rule-based [29] and semantic [37] query optimization work in the sense that a set of rules are utilized to find the best equivalent of an input query (i.e., task execution plans in our case). On the other hand, our optimization algorithm is more flexible to handle the incompleteness and varying reliability of biological data. An optimized task execution plan generates semantically equivalent results (from another data source) to the original task execution plan. However the optimized task execution plan is not guaranteed to produce exactly the same result set with the result of the original task execution plan. Such variations in the results can be desired by biologists since the reliability and performance of biological web sources are not the same.

Among the biological workflow research, Ludascher et al. [7] describes the need for bioinformatics workflow applications on a real world problem (promoter identification) and formalizes basic structure of a workflow system. Addis et al. [10] defines the requirements of a general purpose workflow system. Buttler et al [12] introduces the problem of integrating services from different sources and suggest using semantic web as a resource discovery and data integration solution. IBM BCS-AIS Life Sciences Practice team [24] describes an algebra to analyze workflow systems and develops a workflow system architecture to meet the requirements of a bioinformatics researcher.

Kim et al works on an intelligent assistant workflow construction tool called the Composition Analysis Tool (CAT) [28, 33]. CAT can verify the correctness of a workflow system and it suggests possible ways to extend incomplete workflows. Interactive editing and verification of workflows is out of the scope of this paper.

7. Conclusion

In this paper, we have proposed a task-oriented, toolkit-based integration framework for biological web data sources. And, we have illustrated the resulting issues by describing the construction of the *pathway-infer* system using the functionality provided by the NCBI web site. We have defined and modeled task implementation plan representation. Furthermore, in the context of our running task, we have discussed the optimization of an implementation plan, and experimentally evaluated the optimality of the generated plans. In addition, we have defined a semantic equivalence model for the functions, and formalized the notion of equivalence rules based on a simple grammar. We have also presented a set of axioms enabling the system to infer additional equivalences to be used during the alternative implementation plan discovery process.

8. References

[1] Overbeek, R. et. al., "WIT: Integrated system for high throughput genome sequence analysis and metabolic reconstruction", Nucleic Acids Research, Vol. 28, pp. 123-125.
[2] aMAZE, described at http://www.ebi.ac.uk/research/pfmp
[3] Berkeley BioSpice, online at http://biospice.lbl.gov/home.html.
[4] Jopera system available at, http://www.iks.ethz.ch/jopera
[5] DAML homepage at, http://www.daml.org/
[6] Chawathe, S.S., Abiteboul, S., Widom, J., "Representing and Querying Changes in Semistructured Data", ICDE 1998.
[7] B. Ludäscher, I. Altintas, and A. Gupta. Compiling Abstract Scientific Workflows into Web Service Workflows. In SSDBM'03, July 09-11, 2003
[8] Stein, L., "Integrating biological databases", Nature Reviews Genetics, 4:5, 2003.
[9] Baxevanis, A.D., "TheMolecular Biology Database Collection: 2003 Update", Nucleic Acids Research. 31(1), 1-12, 2003.
[10] Addis M, Ferris J, Greenwood M, Li P, Marvin D, Oinn T, Wipat A. (2003) Experiences with e-Science workflow specification and enactment in bioinformatics. Proc UK e-Science All Hands Meeting 2003, pp. 459-466.
[11] Case Pathways Database System, available at http://nashua.case.edu/pathways.
[12] David Buttler, Matthew Coleman, Terence Critchlow, Renato Fileto, Wei Han, Calton Pu, Daniel Rocco, Li Xiong. Querying Multiple Bioinformatics Information Sources: Can Semantic Web Research Help? SIGMOD Record, Vol 31, No. 4, 2002.
[13] Chen, P.P., "The Entity-Relationship model: toward a unified view of data", ACM Transactions on DataBase Systems, 1:1, 1976.
[14] Jiying Wang, Ji-Rong Wen, Frederick H. Lochovsky, Wei-Ying Ma: Instance-based Schema Matching for Web Databases by Domain-specific Query Probing.VLDB 2004.
[15] Wensheng Wu, Clement T. Yu, AnHai Doan, Weiyi Meng: An Interactive Clustering-based Approach to Integrating Source Query interfaces on the Deep Web. SIGMOD 2004
[16] Hai He, Weiyi Meng, Clement T. Yu, Zonghuan Wu: WISE-Integrator: An Automatic Integrator of Web Search Interfaces for E-Commerce. VLDB 2003: 357-368.
[17] Reactome human pathways project, http://www.reactome.org.
[18] Kyoto Encyclopedia of Genes and Genomes, available at http://www.genome.ad.jp/kegg
[19] Raghu Ramakrishnan, Johannes Gehrke. Database Management Systems. WCB/McGraw-Hill 2004
[20] Zhang Z., He B., Chang K.: Understanding Web Query Interfaces: Best-Effort Parsing with Hidden Syntax. SIGMOD 2004
[21] Michal, G., Biochemical Pathways, Wiley & Sons Inc., 1999.
[22] NCBI E-Utilities, available at http://eutils.ncbi.nlm.nih.gov.
[23] NCBI ESearch web services, available at http://eutils.ncbi.nlm.nih.gov/entrez/eutils/soap/eutils.wsdl.
[24] BioWBI, available at http://www.alphaworks.ibm.com/tech/biowbi
[25] gene_info and gene2refseq files, available at ftp://ftp.ncbi.nlm.nih.gov/gene/DATA/.
[26] Thomas Hernandez, Subbarao Kambhampati: Integration of Biological Sources: Current Systems and Challenges Ahead. SIGMOD Record 2004.
[27] Taverna system available at, http://taverna.sourceforge.net/
[28] Jihie Kim, Marc Spraragen, Yolanda Gil: An intelligent assistant for interactive workflow composition. Intelligent User Interfaces 2004
[29] Lane Warshaw, Daniel P. Miranker: Rule-Based Query Optimization, Revisited. CIKM 1999: 267-275
[30] P. Baker, A. Brass, S. Bechhofer, C. Goble, N. Paton, and R. Stevens. TAMBIS: Transparent Access to Multiple Bioinformatics Information Sources. ISMB, 1998.
[31] L.M. Haas et al. DiscoveryLink: A system for integrated access to life sciences data sources. IBM Systems Journal, vol40, 2001.
[32] Buneman P., Davidson S.B., Hart K., Overton C. ,Wong L. A Data Transformation System for Biological Data Sources. VLDB, 1995
[33] Jihie Kim, Yolanda Gil. Towards interactive composition of semantic web services. AAAI Spring Symposium, 2004
[34] Kepler system available at, http://kepler-project.org/
[35] BioPipe system available at, http://biopipe.org/
[36] myGrid project available at, http://www.mygrid.org.uk
[37] Grant J., Gryz J., Minker J, Raschid L.: Semantic Query Optimization for Object Databases. ICDE 1997: 444-453
[38] W3C WSDL reference at, http://www.w3.org/TR/wsdl