Biological Web Services: Integration, Optimization, and Reasoning

Michael Benedikt
University of Oxford
michael.benedikt@cs.ox.ac.uk

Rodrigo Lopez-Serrano
European Bioinformatics Institute
rls@ebi.ac.uk

Efthymia Tsamoura
University of Oxford
efthymia.tsamoura@cs.ox.ac.uk

Abstract

A vast amount of biological data is now available via web services. Yet the usefulness of this data is limited by the difficulty in performing queries that require data spanning multiple services. We overview a platform which offers integrated data access with minimal user awareness. Users pose high-level queries to this platform, and the system applies a combination of reasoning techniques and cost-based optimization to generate an efficient and reliable implementation on top of the services. We briefly explain the platform's reasoning paradigm, which is based on exact reformulation; we then overview the platform's use on a set of bioinformatics sources, and provide some preliminary results concerning its performance.

1 Introduction

As in most areas of modern society, biology has seen an explosion in the quantity and variety of available data. On the web there are an enormous number of resources providing access to datasets of biological interest; hundreds of overlapping bioinformatics data resources have been already listed by 2013. The predominant means for exposing this data is via web services, generally those abiding by the REST (Representational state transfer) paradigm. The existence of these public interfaces to biological data can enable the building of a wide variety of new applications, from custom data mining tools to new search interfaces. While many of these new data resources are useful to biologists in isolation, even more applications are enabled when scientists combine information from multiple web-based resources. For example, by combining expression data with pathway information biologists can analyse changes in metabolic and signalling processes in cancer diseases, or understand protein disorders through comparative genomics and genetic interactions.

Unfortunately it is difficult for biologists to exploit data from these resources. The tools available to them today include hand-crafted scripts (e.g. in python or perl) and workflow management systems that allow scientists to glue together modules in a component-based way. The key problem in existing techniques for accessing web datasources is that scientists are not isolated from the details of the data resources. This lack of isolation makes it difficult for them to create a process that answers the query by making use of the services (below, we refer to such a process as a "plan").

Although bioinformatics resources can often be accessed via RESTful services, users still have to grapple with a great diversity in the underlying technologies exposed via the services. Some services provide a thin wrapper on top of a traditional database API. Other resources provide keyword-based interfaces. Still others provide navigational interfaces. Many of these interfaces impose additional access restrictions — e.g. certain fields being required, or a limit on the number of requests or amount of results that can be retrieved. Users may not even be aware of the restrictions that are present, but they can impact the results of service calls.

Example 1. *Alice is interested in publications that reference compounds met in humans. She decides to pull publication and compound data from the* PubMed *and* ChEBI *web databases.* PubMed *provides a webservice interface allowing SQL-like interface to access publications. Examples of terms with which you can access* PubMed *publications include the organisms or the compounds referenced in the publication, the publication identifier and the publication year.* ChEBI *provides two interfaces with which you can access compounds: a text interface which returns the identifiers of the compounds that relate to the input text and an identifier-based interface that returns all the data (e.g., the molecular structure) associated with the input compound identifier. The text interface restricts the number of returned compound identifiers to 500.*

Plans that answer Alice's query are not obvious: one first calls the PubMed *interface with input "Human"and then calls the* ChEBI *identifier interface with input the compound identifier found within each returned publication entry. Starting in the opposite way (first calling* ChEBI *and then* PubMed*) will only give an incomplete answer, due to the limitations in the* ChEBI *interface.*

The diversity in interfaces may not only make it difficult to come up with any correct plan, it may make it hard to get a plan that performs well.

Example 2. *Alice is interested in 2015 publications that reference bioassays.* ChEMBL *provides an interface with which users can access all bioassays without providing any*

input It also allows users to do a lookup of a bioassay using the PubMed *identifier of the publication that references this bioassay. One plan to answer the query is to call the input-free interface of* ChEMBL *and then for each returned bioassay to call the* PubMed *interface using the bioassay publication identifier and the publication year (2015). This plan will do in total 1,148,942 requests, the number of bioassays in* ChEMBL*. Another possibility is to call the* PubMed *interface with input 2015 and then for each returned publication to call the* ChEMBL *interface with input the returned publication identifier. This plan will do at most 585,750 requests as there are in total 585,750 publications in* PubMed *published in 2015 and not every 2015 publication references a bioassay.*

Translating a user query into an efficient plan can certainly be done manually. However, while traditional bioinformatics resources hosted locally within a laboratory are quite stable, the velocity of web-based bioinformatics data substantially complicates the development of a plan. Not only do the interfaces characteristics of these resources change over time, but their performance characteristics change as well.

The problem of data integration for biology is by no means a new one — indeed, it has been the subject of intense study within the artificial intelligence, data management, and bioinformatics communities. But there are new technologies that make it timely to make a fresh attack now. First, some of the protocol heterogeneity has lessened, as more and more resources are available via the same protocol. Secondly, much more meta-data is available, in the form of biological ontologies, such as the Experimental Factor Ontology (see `www.ebi.ac.uk/efo`) and the Gene Ontology (see `geneontology.org`). Lastly, there has been much progress in reasoning systems that can support declarative integration.

In this paper we consider an integration system that addresses these challenges. FIBRes (Framework for Integrating Biological REStful services) allows declarative access to a number of biological resources available via web services. It exposes a unified interface — i.e. a "global schema" — via SQL, and implements user queries sent over the interface on top of the services. It differs from prior systems for biological data integration and for web service integration in both the flexibility of its architecture and in the models of integration it supports. The first distinction allows FIBRes to be used in the presence of a variety of constraint and mapping languages. The second distinction give FIBRes advantages in performing cost-based optimization of the corresponding middleware plans. This is particularly relevant in the biological domain, where distinct plans may vary dramatically in performance, as our experiments show.

Organization. Section 2 provides more context for our work, including an overview of prior research. Section 3 explains the FIBRes system at a high level. Section 4 overviews our prototype implementation on top of data from the European Bioinformatics Institute (EBI), one of the leading public providers of biological data, and provides some preliminary experimental results. Section 5 discusses open issues and ongoing work on the system.

2 A brief history of biological data integration

Integration of biological data has been an active research topic for decades. An enormous body of research work has appeared, and a number of excellent surveys are available [Thiam Yui *et al.*, 2011; Gomez-Cabrero *et al.*, 2014; Goble and Stevens, 2008; Paton, 2008; Hernandez and Kambhampati, 2004; Lapatas *et al.*, 2015]. We give a brief overview of some major themes.

Work in the area can be classified within a number of dimensions, including:

- Source modelling.
  Many toolsets focus on sources providing uniform interfaces in a particular data model, such as relational data [Zhang *et al.*, 2011] or nested relations [Davidson *et al.*, 2001]. Others employ a finer specification concerning the *querying capabilities* of sources [Haas *et al.*, 2001; Kambhampati *et al.*, 2004; Thakkar *et al.*, 2005]. Query capability specifications can include whether the source provides keyed lookup facilities, full query language access, or something in between.
- Procedural vs. declarative approaches.
  Some toolsets consist of procedural languages for integration coupled with libraries for certain biological tasks. *Workflow systems*, for example, provide either explicit scripting languages or visual environments for developing applications that access biological data. They deal with the problem of building scientific applications out of components. They are not concerned primarily with querying data, but in allowing users to glue together programs that process data. Taverna [Wolstencroft *et al.*, 2013] is a general-purpose workflow language that has been applied extensively within biology. Galaxy [Goecks *et al.*, 2010] is a workflow system geared specifically towards biology.
  At the other extreme there are approaches that rely throughout on declarative languages — both for enduser access to the data and for specification of the relationships between data items. A prime example of the declarative approach was TAMBIS [Goble *et al.*, 2001] a long-running project for biological data integration utilizing ontologies to express the global schema.
- Relationship of integrated schema and source schemas.
  Within declarative approaches, a further distinction concerns how much indirection is allowed between the integrated schema and the backend sources. Some toolsets focus primarily on a simple kind of federation, providing a single-point of access to multiple biological datasets, allowing them to appear as a single database, but one whose schema is simply the union of all tables in each backend source. This insulates endusers from dealing with many of the issues in merging data (e.g. performing joins across sources), but still require them to understand the details of individual schemas, along with relationships between data in schemas. Bio-Kleisli [Davidson *et al.*, 2001] was an early system for federated access to biological data that follows this model. Chem2Bio2RDF [Chen *et al.*, 2010] is a more recent system that supports querying an interlinked model, but without infer-

ring query results using reasoning.
Other tools expose a global schema that has a more complex relationship with sources. More sophisticated relationships between global schema and stored data are supported by many tools, based on *declarative mappings*, logical constraints relating global and local. The latter approach allows for much more insulation for users, albeit at the cost of more complex database administration, including formation and maintenance of mappings.

- Target implementation and optimizations.
Tools differ in the kind of "integration plans" they generate and the optimizations that can be performed on plans. Discoverylink [Haas *et al.*, 2001] generated plans on top of the Garlic middleware system, which came with a sophisticated optimizer working within a plug-in architecture. Other tools came with middlware targeting particular classes of resources. For example, Thakkar et al. [Thakkar *et al.*, 2005] is one of the few papers dealing specifically with efficiency issues in declarative data integration on top of biological web services. They make use of the common "certain answer semantics" for querying global schemas defined by powerful declarative mappins (see discussion below). Plans for providing such answers will generally require recursion, and [Thakkar *et al.*, 2005] provides a streaming dataflow engine that can handle recursive queries. They also provide optimizations geared towards reducing the number of web service calls.

FIBRes **in context.** We now place our own work, FIBRes, within these dimensions. In terms of *data model* we view sources as relations equipped with collection of access methods (in the same spirit as [Kambhampati *et al.*, 2004; Thakkar *et al.*, 2005]), which expose look-up style interfaces to the relations. We focus on *purely declarative techniques*, providing SQL access on top of a global schema defined by logic-based integrity constraints relating it to the local sources. Thus our work is completely orthogonal to systems like Taverna or Galaxy, or to declarative systems that do not use reasoning, such as Chem2Bio2RDF. The major distinction of FIBRes from prior work in the logic-based space is *the semantics we consider for implementing a query, and the ability (enabled by our chosen semantics) to do cost-based optimization on web-services*. In terms of semantics, we focus on getting *exact reformulations* of a user query Q: a plan PL making use of the exposed access methods with the property that: for every instance I for the global and local schemas together that satisfies the mapping rules and constraints, PL run over the backend sources in I gives the same result as Q evaluated on I. Exact reformulations are available for *access-determined scenarios* [Benedikt *et al.*, 2016; 2015b]: those where the information available from the interfaces *determines* the output of the query. Access-determinacy always holds in the case of "global-as-view" mappings, where the global schema is defined by a set of queries over the sources: for global-as-view, each instance of the sources defines a single instance of the global schema (see Figure 2, left), and thus any query over the global schema is access-determined. But access-determinacy also holds in many other cases, such as when the global schema has several alternative definitions based on source information, or when the global schema loses information, but that information does not impact the query (see Figure 2, right). Thus the exact reformulation approach is strictly more general than global-as-view. It is less general than approaches such as "local-as-view" [Lenzerini, 2002] which allow each instance of the local schema to correspond to many possible instances of the global schema with different query results, and defines the answer to a query as being the intersection over all results (the "certain answers": see Figure 2, middle). Thus the certain answer semantics is well-defined even when access-determinacy fails. The advantage of exact reformulations over this broader approach is that exact reformulations are generally smaller in size (and in runtime efficiency) and easier to optimize than plans that retrieve the certain answers. For example, exact reformulations never require recursion, and given that recursively probing web services is extremely expensive, the ability to avoid recursion is particularly significant in the domain of web service integration. The use of exact reformulations has *allowed us to adapt cost-based optimization to logic-based data integration of services*, as we explain in Section 3. Further the exact reformulation approach does not require strong restrictions on the constraint or mapping language for capturing the relationships between and among global schema and sources (in contrast, e.g. to work in Ontology Based Data Access). To our knowledge, FIBRes is the *only* system for logic-based relational and web-service data integration with cost-based optimization.

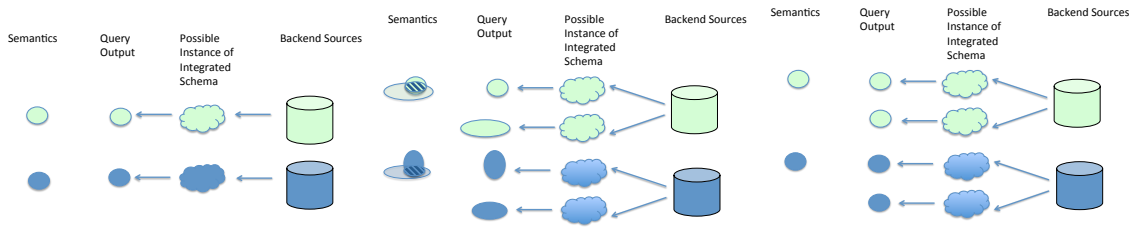
3 FIBRes architecture

FIBRes is built on top of the PDQ system for data integration [Benedikt *et al.*, 2014; 2015a]. The system's architecture is the depicted in Figure 1, and features a meta-data manager, a planner, a runtime, and wrappers.

Metadata. FIBRes requires metadata about both backend sources and the integrated "global schema". For the backend sources, the metadata consists of the interfaces that each source supports, specified as functions that take a fixed set of key-values (the *inputs* of the interface) and returning a list of tuples. The idea is that by binding the inputs to values, the web service can be invoked to get all matching tuples. We say that the interface is *accessed* with that binding.

For the global schema, FIBRes requires a collection of tables with their attributes and data types, and a collection of *integrity constraints*. The constraints can include mapping rules that relate backend sources to global schema tables, as well as relationships and invariants that hold within the global schema or among the sources. Although the FIBRes architecture is quite flexible about the constraint language, the current implementation of FIBRes supports only *dependencies*: either tuple-generating dependencies (TGDs) or equality-generating dependencies (EGDs) [Fagin *et al.*, 2005].

Queries and plans. The main function of FIBRes is to take a *user query* and search for a *plan* that is an exact reformulation of the query. User queries are specified as SQL basic SELECT queries over the global schema, while plans



(a) Global as view (b) General mappings (c) Access-determined

are a sequence of access commands and middleware data manipulation commands. Commands refer to a set of *temporary tables* that are maintained in the middleware. An access command takes an interface and the contents of one temporary table and performs a "bulk access" — accessing the interface with every tuple in the table, putting the outputs in another temporary table. A middleware data manipulation command performs standard database operations (e.g. in relational algebra) on temporary tables. Thus a join across sources S_1 and S_2 could be implemented by issuing access commands on each source, putting the results in temporary tables T_1 and T_2, and then performing a middleware command that joins T_1 and T_2.

Planner. The planner is the central object in the architecture, taking a query and searching for an appropriate plan PL. This is done by reasoning with the dependencies. For each query Q and set of dependencies Σ, we can come up with a query Q' and an additional set of constraints Σ' such that: Q has an exact reformulation iff the *entailment*

$$Q \wedge \Sigma \models (\Sigma' \rightarrow Q')$$

holds. Informally, the entailment above represents a proof that the information in the interfaces determines the output of the query Q; a proof that we have the picture in Figure 2, right, rather than that in Figure 2 center.

Furthermore, for every proof of the entailment, one can extract a corresponding plan that is an exact reformulation of Q. The planning module searches for a proof. If no such proof exists, this means that the query has *no exact reformulation*. For example, in a variation of Example 1 in which all of the interfaces can only return at most one tuple, Alice's query would not have any exact reformulation, and the system will report this. If there are many proofs, then the planner searches through all of them using the cost of the corresponding plan to guide the search. In the case where the constraints Σ consist of TGDs and EGDs, the same is true for Σ', and thus proofs of the entailment $Q \wedge \Sigma \models (\Sigma' \rightarrow Q')$ can be generated using the well-known forward-chaining algorithm known as *the chase* [Maier *et al.*, 1979; Fagin *et al.*, 2005; Onet, 2013].

The *proof-to-plan* module is in charge of building execution plans from proofs, making calls to the reasoner for consequence closure. The cost module evaluates and compare the quality of our plans. Cost for access commands is assumed to be proportional to the number of bindings to a given access method, multiplied by a per-method constant. In the absence of any information on the performance of methods, each method is given the same per-method cost.

Above we have been very high-level. The details of the transformation from Q, Σ to Q', Σ', the plan-extraction algorithm, and the proof calculus, can be found in a [Benedikt *et al.*, 2015b; 2016]. But for the purposes of what follows, the important take-away is: *we search through a space of proofs, with each proof validating that we have sufficient accessible data to answer the query; each proof corresponds to a path of information requests, and thus in searching the proofs we are searching through the corresponding sequences of access methods that can answer the user query. As we search we use cost to prune the space of alternative proofs and plans to explore.* In Example 2 one proof would correspond to the plan first making input-free access to ChEMBL and then using the results in PubMed, while a second proof would correspond to the plan first accessing PubMed with input 2015 and using the results to access ChEMBL.

Cost. FIBRes supports a variety of cost functions. For example, middleware operations can be estimated using standard "textbook" formulas (e.g. [Ramakrishnan and Gehrke, 2003], Chapters 12 and 14) in terms of an estimate on the size of the output. In our prototype we only consider the cost of access commands, ignoring the processing of data in middleware. For each access command we estimate its cost based on the average number of tuples returned, which is estimated from sampling the data. In Example 2 we use our estimates of the selectivity of accesses to ChEMBL and PubMed to determine that the second plan does fewer web service calls, and hence is to be preferred.

Runtime. FIBRes has an execution environment for evaluating plans. Operators are evaluated in pipelined fashion, with intermediate results from accesses and relational operators being buffered in the middleware. To deal with transient web-service failures in access commands, the middleware has a parameterized restart threshold, after which it re-initiates an access by reconnecting to the service.

Wrappers. The wrapper layer acts as an interface with the services, containing any service-specific information (destination urls, service parameters).

4 Initial data sets and results

Our initial dataset included data from a variety of biological datasources, all obtained via the interfaces of the European Bioinformatics Institute. We chose to integrate data from these resources as they all support a RESTful access API and expose relational data.

- ChEMBL is a bioassay database [Bento *et al.*, 2014]. The web services in ChEMBL implement SQL-like

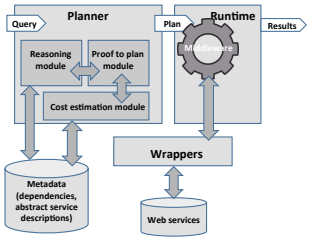


Figure 1: System's architecture.

access interfaces on top of which users can submit conjunctive queries over a single table. We considered the `Activity`, `Assay`, `Document`, `Molecule`, `Target` and `TargetComponent` web services. More details about ChEMBL web services can be found under `https://www.ebi.ac.uk/chembl/ws`.

- UniProtKB [The UniProt Consortium, 2015] provides functional information on proteins. It consists of two different datasets, Swiss-Prot and TrEMBL, where Swiss-Prot is a datasource of manually annotated proteins, and TrEMBL is an automatically-annotated supplement. The data in UniProtKB is available to users both through services that allow key-based lookups and through services that implement SQL-like accesses. See `http://www.uniprot.org/help/programmatic_access`.
- Reactome [Milacic *et al.*, 2012] is a pathway database. We integrated the Reactome web services `frontPageItems`, `queryById` and `speciesList` (see `http://reactomews.oicr.on.ca:8080/ReactomeRESTfulAPI/ReactomeRESTfulAPI.html`). `frontPageItems` takes as input a species name and returns all the pathways that relate to this species. `queryById` allows key-based lookups to species and pathways, while `speciesList` allows input-free access to the species that is found within Reactome.
- EuropePMC [Europe PMC Consortium, 2015] provides access to abstracts and full text of the biomedical literature from life science journals, online books, and other resources including PubMed. We integrated several web services from EuropePMC to pull metadata about articles, citations, and references. `search` performs SQL-like searches over the available publications and it is parametrised with the type of returned metadata: when users specify the option `idlist` the web service returns the list of identifiers and sources of the publications that match the input SQL query. The option `lite` returns additional metadata about the queried publications including the publication authors, title, journal, volume and year. Finally, `citations` and `references` allow identifier-based searches over the publications that cite (are referenced by) a given article. More details about EuropePMC web services can be found under `http://europepmc.org/restfulwebservice`.

In addition, we made use of an aggregate interface from EBI, EB-eye [Squizzato *et al.*, 2015]. This is a text search engine that provides access to EBI's data resources through a RESTful API. We used EB-eye's REST interface to access data from ChEMBL and UniProtKB. Note that the EB-eye web service returns a subset of the data that is available through the web services provided by the different resource. See `http://www.ebi.ac.uk/Tools/webservices/services/eb-eye_rest` for a description of the RESTful interface.

FIBRes models web services with an SQL-like search interface as access methods; each access is translated to an SQL command consisting of a single conjunct. We had to restrict the data that is pulled per web service call; as FIBRes only integrates data in relational format, we do not parse the web service attributes that may be populated by a list of values. Most of the web services that are integrated are parametrised with the preferred page size and output format (e.g., JSON or XML). The page size, was fixed to be the maximum value allowed by the resource; for the output format, we preferred JSON if available; otherwise XML.

Global schema. We created a global schema, mappings, and constraints manually. The global schema consists of a set of views that represent the data of the remote datasources. The objective is to conceal the heterogeneity of the remote datasources and the different access interfaces, providing, at the same time, a simple view of the data.

To take the data in ChEMBL as an example, we created one view for each web service in this source. That is, we created views $V_{Activity}$, V_{Assay}, $V_{Document}$, $V_{Molecule}$, V_{Target} and $V_{TargetComponent}$. The schema of each view is the output schema of the corresponding web service, modulo some normalization to change multi-valued attributes to multiple scalar attributes. We adopted a similar rationale to create views for the data of the remaining datasources. The view $V_{Protein}$ captures the data returned by UniProtKB web services and its schema is the output schema of the corresponding services. We also defined the view $V_{Pathway}$ with schema the output schema of the `frontPageItems` and `queryById` web services. In order to represent data in EuropePMC, we defined the views $V_{Publication}$, $V_{Citation}$ and $V_{Reference}$. $V_{Publication}$ captures the data returned by `search`, while $V_{Citation}$ and $V_{Reference}$ capture the data returned by `citations` and `references`, respectively. Finally, we defined bidirectional mappings between the views created out of the ChEMBL and UniProtKB web services and EB-eye's web service. Figure 2 shows a portion of the global schema.

Note that there is a many-to-one relationship between data sources and virtual tables in the global schema. The redundancy in data resulted in the existence of multiple plans for all of our user queries.

Results. We report results on the performance of FIBRes on 21 sample queries that were manually created. They include examples representing the tasks in Example 2 [1] as well as other queries that span multiple resources.

[1] ChEBI is not yet modeled in our system, so Example 1 is not in our benchmark

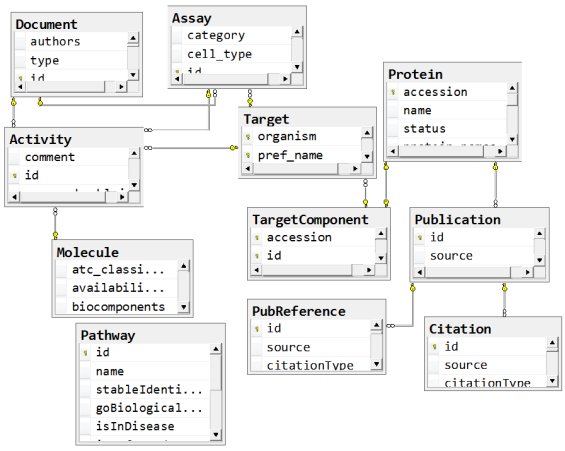


Figure 2: Global schema

We report for each query the time spent by the plan generated by FIBRes, comparing it with the time spent by a "randomly generated plan". To approximate a randomly generated plan, we ran FIBRes with no cost preference, and generated multiple correct plans: we then chose one of the plans randomly. For each query we report the time to get the first result tuple and the time to get the first 500 result tuples (in milliseconds). Note that some queries return no tuples (Q8, Q9, Q17). Q20 returns only about 300 tuples, but getting the complete answer set is infeasible for the random plan.

We see that FIBRes-generated plans outperform the random ones. The large gaps in times illustrate the wide range of plans that are available for these queries.

	Time to first tuple		**Time to 500 tuples**	
	FIBRes	**Random**	FIBRes	**Random**
Q1	912	1053	24771	31583
Q2	1753	2176	45922	67241
Q3	1810	25641	26801	27351
Q4	28083	69934	30414	90141
Q5	23529	70874	23547	90141
Q6	5743	12447	8150	26762
Q7	3403	1318997	923845	> 30min
Q8	27574	> 30min	27574	> 30min
Q9	21351	> 30min	21351	> 30min
Q10	30399	1437083	61075	> 30min
Q11	47555	1714716	> 30min	> 30min
Q12	4542	9413	19777	674909
Q13	39189	41687	40519	42767
Q14	2584	6583	685652	840745
Q15	568463	> 30min	855158	> 30min
Q16	29514	> 30min	198677	> 30min
Q17	91310	> 30min	91310	> 30min
Q18	90941	110235	96589	111646
Q19	117499	> 30min	200158	> 30min
Q20	6567	> 30min	1978712	> 30min
Q21	36167	45037	170909	> 30min

Figure 3: FIBRes vs. random reformulations

The time to run the planner for these queries can be non-trivial, ranging from about 1 sec to several minutes, due to the large number of plans that need to be considered — details can be found in the technical report. But note that this time is often orders of magnitude smaller than the gap between runtimes shown in Figure 3. Further, the expectation is that planning need only be done once, while the gains in runtime are exploited repeatedly over many runs of the plan.

5 Conclusions and Ongoing work

Our current FIBRes prototype is only a small step towards declarative integration of web-service hosted biological data. We claim only that the current results show some promise in devising a system that searches through multiple exact reformulations to find an efficient plan.

Some of the major ongoing activities in improving FIBRes are as follows:

- **Incorporating query-based access.** FIBRes supports typical key-based lookup services, where the arguments are values for a fixed set of attributes. Some biological web services permit a more powerful interface, allowing arbitrary attribute equalities on a given relation. It is easy to support such interfaces at the planning level, but accounting for the trade-offs between sending numerous restrictive queries versus a single broad query is challenging. We are currently extending our cost-estimation procedures to account for these trade-offs.
- **Incorporating keyword-based access.** An important trend in biological data management is to store the underlying data in a document store, such as Lucene, and provide a web service layer that wraps a keyword-based interface. Typically a document store will return the top K matches according to a particular scoring function, where the scoring function may either be a standard IR measure (e.g. TF/IDF) or a user-defined function. Accounting for top-K semantics requires extensive changes in the FIBRes architecture and optimizer.
- **Multi-planning for reliability.** Web services often fail, and thus an important component of an integration system is facilities for identifying and reacting to failure. As mentioned in Section 3, FIBRes has support for handling *transient failures*, such as arise when a service is temporarily unreachable or (more commonly) when a service responds to a large number of requests by throttling the client. FIBRes currently performs static planning, returning a single plan that interacts with web-services. This approach is clearly brittle in the presence of long-duration service failures. We are currently modifying the planning algorithm to produce alternate plans that allow resilience in the presence of failures of any single service. The resulting plan would be coupled with a performance monitor that manages failover. The knowledge of semantic relationships between services plays a key role here; it allows FIBRes to devise alternate plans that are widely different syntactically, but which provide the same answers

  Going beyond this, we are incorporating *dynamic re-planning*, allowing the planner to re-plan as new performance statistics arrive.

References

[Benedikt *et al.*, 2014] M. Benedikt, J. Leblay, and E. Tsamoura. PDQ: Proof-driven query answering over web-based data. In *VLDB*, 2014.

[Benedikt *et al.*, 2015a] M. Benedikt, J. Leblay, and E. Tsamoura. Querying with access patterns and integrity constraints. In *VLDB*, 2015.

[Benedikt *et al.*, 2015b] M. Benedikt, B. ten Cate, and E. Tsamoura. Generating plans from proofs. In *TODS*, 2015.

[Benedikt *et al.*, 2016] M. Benedikt, J. Leblay, B. ten Cate, and E. Tsamoura. *Generating Plans from Proofs: The Interpolation-based Approach to Query Reformulation*. Morgan & Claypool, 2016.

[Bento *et al.*, 2014] P. Bento, A. Gaulton, A. Hersey, L. J. Bellis, J. Chambers, M. Davies, F. A. Krger, Y. Light, L. Mak, S. McGlinchey, M. Nowotka, G. Papadatos, R. Santos, and J. P. Overington. The ChEMBL bioactivity database: an update. *Nuc. acids research*, 42, 2014.

[Chen *et al.*, 2010] B. Chen, X. Dong, D. Jiao, H. Wang, Q. Zhu, Y. Ding, and D. J. Wild. Chem2bio2rdf: a semantic framework for linking and data mining chemogenomic and systems chemical biology data. *BMC Bioinformatics*, 11(1):1–13, 2010.

[Davidson *et al.*, 2001] S. B. Davidson, J. Crabtree, B. P. Brunk, J. Schug, V. Tannen, G. C. Overton, and C. J. Stoeckert Jr. K2/Kleisli and GUS: Experiments in integrated access to genomic data sources. *IBM Systems Journal*, 40(2):512–531, 2001.

[Europe PMC Consortium, 2015] Europe PMC Consortium. Europe PMC: a full-text literature database for the life sciences and platform for innovation. *Nuc. acids research*, 43, 2015.

[Fagin *et al.*, 2005] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. *Theoretical Computer Science*, 336(1):89–124, 2005.

[Goble and Stevens, 2008] C. A. Goble and R. Stevens. State of the nation in data integration for bioinformatics. *Journal of Biomedical Informatics*, 41(5):687 – 693, 2008.

[Goble *et al.*, 2001] C. A. Goble, R. Stevens, G. Ng, S. Bechhofer, N. W. Paton, P. G. Baker, M. Peim, and A. Brass. Transparent access to multiple bioinformatics information sources. *IBM Systems Journal*, 40(2):532–551, 2001.

[Goecks *et al.*, 2010] J. Goecks, A. Nekrutenko, J. Taylor, and The Galaxy Team. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biol*, 11(8):R86, 2010.

[Gomez-Cabrero *et al.*, 2014] D. Gomez-Cabrero, I. Abugessaisa, D. Maier, A. Teschendorff, M. Merkenschlager, A. Gisel, E. Ballestar, E. Bongcam-Rudloff, A. Conesa, and J. Tegnér. Data integration in the era of omics: current and future challenges. *BMC Systems Biology*, 8(2):1–10, 2014.

[Haas *et al.*, 2001] L. M. Haas, P. M. Schwarz, P. Kodali, E. Kotlar, J. E. Rice, and W. C. Swope. Discoverylink: A system for integrated access to life sciences data sources. *IBM Systems Journal*, 40(2):489–511, 2001.

[Hernandez and Kambhampati, 2004] T. Hernandez and S. Kambhampati. Integration of biological sources: Current systems and challenges ahead. *SIGMOD Record*, 33(3):51–60, September 2004.

[Kambhampati *et al.*, 2004] S. Kambhampati, U. Lambrecht, E.and Nambiar, Z. Nie, and G. Senthil. Optimizing recursive information gathering plans in EMERAC. *J.Int. Inf. Sys.*, 22(2):119–153, 2004.

[Lapatas *et al.*, 2015] V. Lapatas, M. Stefanidakis, R. C. Jimenez, A. Via, and M. V. Schneider. Data integration in biological research: an overview. *Journal of Biological Research*, 22(1), 2015.

[Lenzerini, 2002] M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, 2002.

[Maier *et al.*, 1979] D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. *TODS*, 4(4):455–469, 1979.

[Milacic *et al.*, 2012] M. Milacic, R. Haw, K. Rothfels, G. Wu, D. Croft, H. Hermjakob, P. D'Eustachio, and L. Stein. Annotating cancer variants and anti-cancer therapeutics in reactome. *Cancers*, 4(4), 2012.

[Onet, 2013] A. Onet. The chase procedure and its applications in data exchange. In *Data Exchange, Information, and Streams*, pages 1–37, 2013.

[Paton, 2008] N. W. Paton. Data integration in the life sciences: Fun, findings and frustrations. In *DILS*, 2008.

[Ramakrishnan and Gehrke, 2003] R. Ramakrishnan and J. Gehrke. *Database management systems (3. ed.)*. McGraw-Hill, 2003.

[Squizzato *et al.*, 2015] S. Squizzato, Y. M. Park, N. Buso, T. Gur, A. Cowley, W. Li, M. Uludag, S. Pundir, J. A Cham, H. McWilliam, and R. Lopez. The EBI Search engine: providing search and retrieval functionality for biological data from EMBL-EBI. *Nuc. acids research*, 43(W1), July 2015.

[Thakkar *et al.*, 2005] S. Thakkar, J. L. Ambite, and C. A. Knoblock. Composing, optimizing, and executing plans for bioinformatics web services. *VLDB J.*, 14(3):330–353, 2005.

[The UniProt Consortium, 2015] The UniProt Consortium. UniProt: a hub for protein information. *Nuc. Acids Research*, 43(D1), 2015.

[Thiam Yui *et al.*, 2011] C. Thiam Yui, L. J. Liang, W. Jik Soon, and W. Husain. *A Survey on Data Integration in Bioinformatics*. 2011.

[Wolstencroft *et al.*, 2013] K. Wolstencroft, R. Haines, D. Fellows, A. R. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. Nieva de la Hidalga, M. P. Balcazar Vargas, S. Sufi, and C. A. Goble. The Taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. *Nuc. Acids Research*, 41:557–561, 2013.

[Zhang *et al.*, 2011] J. Zhang, S. Haider, J. Baran, A. Cros, J. M. Guberman, J. Hsu, Y. Liang, L. Yao, and A. Kasprzyk. BioMart: a data federation framework for large collaborative projects. *The Journal of Biological Databases and Curation*, 2011.