Using Hierarchical Task Network Planning Techniques to Create Custom Web Search Services over Multiple Biomedical Databases

Miguel García-Remesal

Biomedical Informatics Group, Dep. Inteligencia Artificial, Facultad de Informática, Universidad Politécnica de Madrid. Campus de Montegancedo S/N, 28660 Boadilla del Monte, Madrid, Spain
mgarcia@infomed.dia.fi.upm.es

Abstract. We present a novel method to create complex search services over public online biomedical databases using hierarchical task network planning techniques. In the proposed approach, user queries are regarded as planning tasks (goals), while basic query services provided by the databases correspond to planning operators (POs). Each individual source is then mapped to a set of POs that can be used to process primitive (simple) queries. Advanced search services can be created by defining decomposition methods (DMs). The latter can be regarded as "recipes" that describe how to decompose non-primitive (complex) queries into sets of simpler subqueries following a divide-and-conquer strategy. Query processing proceeds by recursively decomposing non-primitive queries into smaller queries, until primitive queries are reached that can be processed using planning operators. Custom web search services can be created from the generated planners to provide biomedical researchers with valuable tools to process frequent complex queries.

Keywords: Database integration, automated planning, hierarchical task network planning.

1 Introduction

Over the last few years, there has been a dramatic increment in the number of publicly available biomedical databases [1]. The latter provide information on complementary topics, including biomedical literature, diseases, genes, proteins, polymorphisms, etc.

Public online resources are normally focused on single topics. For instance, PubMed [2] provides references to literature, while OMIM [3] is a database of genetic disorders. Therefore, to process frequent queries that involve searching for several topics, users are required to manually follow ad-hoc search flows that define chained sequences of searches on different databases. An example of such queries would be "retrieve all European laboratories that perform diagnostic tests for the Cystic Fibrosis disease". This search would entail the use of the following search flow: i) searching the OMIM database to get the disease identifier (MIM ID) associated to that particular disease, and ii) querying the EDDNAL [4] database using the previously obtained MIM ID to retrieve all the laboratories that perform clinical tests for the target genetic disorder. Manually executing these search flows becomes a harder task when the number of involved databases increases. Therefore, there is a need for novel methods and tools to automatically perform these complex searches.

In this paper we propose the use of hierarchical task planning (HTN) techniques [5, 6] to facilitate the creation of complex query services over public online biomedical databases. In our approach, user queries are regarded as tasks to be performed—i.e. goals to be achieved—while basic query services provided by the sources are viewed as primitive planning operators (POs). For instance, the PubMed biomedical literature database provides different basic query services—e.g. searches by topics, authors, journals, etc. Thus, a user query such as "find all articles by John Doe" could be considered as a task (or goal) to be achieved. Similarly, the query service "search all the articles by a concrete author" provided by PubMed could be regarded as a PO that can be used to carry out that particular task.

The idea behind the proposed method is exploiting the domain knowledge provided by the search flows followed by users to create query decomposition methods (DMs) to deal with non-primitive queries. The latter are complex queries that involve a chained sequence of searches in one or more databases. Unlike simple queries, non primitive queries cannot be processed by applying a single primitive planning operator. Instead, they require the execution of a sorted sequence of primitive operators. Hence, DMs can be defined as pieces of domain knowledge that describe how to recursively decompose complex queries into a set of simpler queries following a divide-and-conquer strategy. These subqueries can be either primitive or non-primitive. Further decomposition is then performed on non-primitive subqueries until primitive queries are reached that can be processed using individual planning operators.

Once a planner based on specific DMs extracted from a given search flow has been created, it can be used as the core of a web search service that supports that concrete search flow. The generated web services can be either used as independent tools, or can be reused as building blocks to create more complex query services.

This paper is organized as follows. In the next section, we focus on the methods we used to create web search services from search flows using HTN planning techniques. Next, we present the results of an experiment that we conducted to test our approach. The experiment involved an actual search flow frequently used in the domain of genetic diseases. After that, we briefly compare our method with other similar approaches. Finally, we draw the conclusions.

2 Methods

The first step toward the creation of custom web services supporting concrete search flows is the identification of the involved databases. Once the sources have been identified, each database is then mapped to a set of planning operators (POs) describing the basic search services provided by that particular source. To represent these operators, we use the classical representation [7] for planning problems, based on first order logic (FOL).

A PO can be defined as a tuple *o* = (name(*o*), pre(*o*), del(*o*), add(*o*)) whose elements are as follows.

- name(*o*) represents the name of the operator. It is a syntactic expression of the form n(x_1: t_1, x_2: t_2, …, x_k: t_k), where n is a unique operator symbol, and x_1, …, x_k are all variable symbols that represent the operator's parameters. The symbols t_1, …, t_k represent the types associated to the different variables. Operators can be instantiated by binding one or more variables in name(*o*) to constant values belonging to their corresponding types.
- pre(*o*) is a set of literals—i.e. FOL atoms—that represent the precondition of the operator *o*. An instance of *o* is said to be applicable iff its precondition holds in the current state.
- del(*o*) is the set of negative effects of the operator, also called the *deletion list*. It is a set that includes all the atoms that will no longer hold after the execution of the operator.
- add(*o*) is the set of positive effects of the operator, also called the *addition list*. It contains all the atoms that will hold after the execution of the operator.

To illustrate the translation of basic query services into POs, let us consider the PubMed database. As stated previously, PubMed provides the service "search all the articles by a given author". The PO associated to this query service can be defined as follows.

PUBMED_QUERY_OP_search_by_author(?q: string)

pre: AUTHOR_QUERY_STRING(?q)
del: AUTHOR_QUERY_STRING(?q)
add: {article_by(?p ?a) | paper ?p is authored by author ?a}

As shown above, the operator *PUBMED_QUERY_OP_search_by_author* takes as input the variable *?q* of type *string*. Using this parameter we indicate the author of the papers that we are interested in retrieving. Regarding the operator's precondition, it states that the atom *AUTHOR_QUERY_STRING(?q)* must hold in the current state for the operator to be applicable. This atom is automatically asserted when the user launches the query. After the search has been completed, the atom *AUTHOR_QUERY_STRING(?q)* is no longer needed, and thus, it is discarded. Besides, a set of instances of the atom *article_by(?p ?a)* are automatically asserted, thus holding in the next state. This set includes all instances of the atom *article_by(?p ?a)* such that the paper *?p* has been published by the author *?a* according to the records retrieved from PubMed. Note that more than one autor *?a* might match the user query *?q*.

Apart from creating operators specifically designed to process user queries, it is also necessary defining POs that enable the planning algorithm to achieve intermediate goals. An example is provided next.

PUBMED_OP_search_by_author(?pn: person)

pre: person(?pn)
del: person(?pn)
add: {article_by(?p ?pn) | paper ?p is authored by author ?pn}

The PO shown in the above example is similar to the operator *PUBMED_QUERY_OP_search_by_author*. Indeed, both POs provide the same functionality. The only difference is that the operator's precondition must be asserted by a previously applied operator rather than by a user query.

Once all the target databases have been mapped to sets of primitive POs, it is now possible to encode a query processing task as a planning problem.

The corresponding planning problem is defined by the tuple $P(O, s_0, g)$, whose elements are as follows.

- O is a set of operators. This includes all POs provided by the involved sources.
- s_0 is the initial state of the world. It is represented by a set that includes all the atoms corresponding to the query launched by the user. For instance, the user query *"retrieve all articles authored by J. Doe"* would be represented by the initial state s_0 = *{AUTHOR_QUERY_STRING("J. Doe")}*.
- g is the goal state. It is represented by a set that includes all atoms that must hold in the goal state. Going back to the previous example, the goal represented by the set g = *{article_by(?a "J. Doe")}* includes all instances of the predicate *article_by* such that "J. Doe" is the author of the article *?a*.

Once the planning problem has been defined, it is now possible to extract a solution plan $\Pi = \{\pi_1, \pi_2, \ldots, \pi_m\}$ using any automated planning method [7]. Note that each $\pi_i \in \Pi$ represents a search in one concrete database that takes as input some of the results provided by previously executed searches—i.e. $\pi_{1, \ldots,} \pi_{i-1}$.Thus, Π represents a query execution plan that can be automatically executed.

The main drawback of this approach is that classical planning techniques do not exploit the expert knowledge provided by the query flows followed by users to manually execute complex frequent queries.

Among all available automated planning techniques, we believe that hierarchical task network planning techniques (HTN) [5, 6] are the best suited to address the creation of custom web search services. This is partly because HTN provides specific artifacts called decomposition methods (DMs) that can be regarded as "recipes" that describe how a human expert may think about manually processing a complex query.

Hence, we propose translating the query flows used by human experts to process frequent queries into DMs built upon primitive operators provided by the databases. These DMs i) facilitate the planning task, and ii) generate web search services that are similar to how human experts manually execute the queries.

DMs can be regarded as methods to solve complex tasks that can be recursively divided into sets of simpler tasks following a divide-and-conquer strategy. For instance, the query "find all European laboratories that perform diagnostic tests for the Cystic Fibrosis disease" is handled by human experts by decomposing it into two simpler queries. First, the user queries the OMIM database to obtain the disease identifier (MIM ID) associated to the target disease (i.e. Cystic Fibrosis). Next, the EDDNAL database is searched by providing the previously retrieved MIM ID as input. This produces a result set including all European laboratories that perform genetic tests for the Cystic Fibrosis genetic disorder. This search flow can be easily translated into a set of DMs and primitive operators as shown below.

METHOD_search_for_lab_by_disease(?q: string)
task: search_for_lab_by_disease(?q: string)
pre: none
del: none
subtasks: <OMIM_QUERY_OP_get_matched_diseases(?q), get_laboratories(?m)>

METHOD_get_laboratories()
task: get_laboratories()
pre: none
del: none
subtasks: <EDDNAL_OP_get_laboratory(?m), get_laboratories()>

OMIM_QUERY_OP_get_matched_diseases(?q: string)
pre: DISEASE_QUERY_STRING(?q)
del: DISEASE_QUERY_STRING(?q)
add: {disease-id(?m ?d) | ?m is the identifier associated to disease[1] ?d according to OMIM}

EDDNAL_OP_get_laboratories(?m: disease-id)
pre: disease-id(?m ?d)
del: disease-id(?m ?d)
add: {test-lab(?l ?m ?d) | ?l is a lab that performs tests for disease ?d according to EDDNAL}

When using HTN techniques, goals are no longer represented as sets of atoms that must hold in the goal state. Instead, goals are regarded as lists of tasks to be performed. Thus, the statement of a HTN planning problem is represented by the tuple $P = (O, M, s_0, t)$, where M is a set that includes all the DMs, t is a list of tasks to be achieved, and O and s_0 have the same meaning as in classical planning.

Using the above definitions, the query flow "retrieve all European laboratories that perform diagnostic tests for the Cystic Fibrosis disease" can be stated as the following planning problem *P = {O, M, s_0 = {DISEASE_QUERY_STRING("Cystic Fibrosis")}, t = <search_for_lab_by_disease(?q)>}.*

This HTN planning problem can be solved using the SHOP2 HTN planning algorithm [8]. SHOP2 proceeds by recursively searching for a suitable method or operator to achieve the goal task. In the previous example, the task *search_for_lab_by_disease* can be performed by applying the method *METHOD_search_for_lab_by_disease*. Once the method has been applied, the task *search_for_lab_by_disease* is replaced with the corresponding list of subtasks specified by selected method, i.e. *<OMIM_QUERY_OP_get_matched_diseases, get_laboratories(?m)>*. Note that the first subtask can be directly achieved by a primitive operator, while, the second is a non-primitive subtask that requires further decomposition using a suitable method—i.e. the method *METHOD_get_laboratories*.

[1] Note that more than one disease *?d* might match the query string *?q*.

Once the search flow has been converted into a HTN planning problem, we then use a modified version of the JSHOP2 planner generator [9] to create a web search service. The modified planner generator takes as input the formal definition of the HTN problem augmented with additional information. The latter includes directions on how to enter and extract information from the web pages belonging to the different databases. The planner generated by JSHOP2 is then used as the core of the newly created web service, acting as a mediator responsible for query processing.

3 Results

In this section, we present the results of an experiment that we carried out to test the proposed approach. The experiment involved a complex search flow frequently used by biomedical researchers on the area of genetic diseases. The search flow is depicted in the figure below.

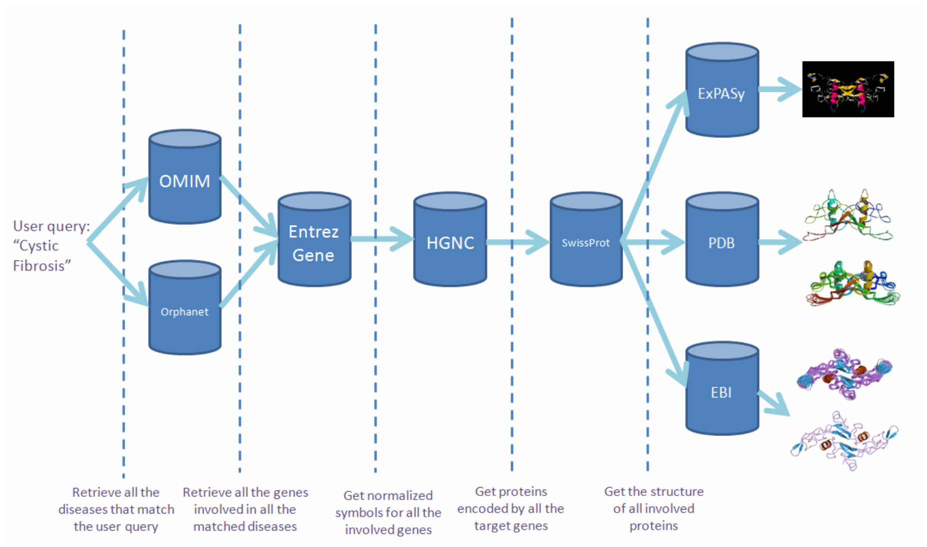
User query: "Cystic Fibrosis"
OMIM
Orphanet
Entrez Gene
HGNC
SwissProt
ExPASy
PDB
EBI
Retrieve all the diseases that match the user query
Retrieve all the genes involved in all the matched diseases
Get normalized symbols for all the involved genes
Get proteins encoded by all the target genes
Get the structure of all involved proteins

Fig. 1. Overview of the search flow involved in the experiment

As shown in figure 1, the search flow is aimed to retrieve the three-dimensional structure of all proteins involved in a given genetic disease. This generic query entails a complex chained search over 8 databases focused on different topics. These topics include diseases (OMIM [3]), rare genetic diseases (Orphanet [16]), genes (Entrez Gene [17]), normalized gene symbols (HGNC [18]), proteins (SwissProt [19]), and protein structures (ExPASy [19], PDB [20] and EBI [21]). The different phases of the search are shown in the figure.

We encoded the search flow as an equivalent HTN planning problem using the methods described in the previous section. The obtained HTN problem included 8 POs and 5 DMs.

Once the search flow was encoded as a HTN planning problem, we used the JSHOP2 planner generator to automatically create a HTN planner implementing the target search flow. The generated planner was then encapsulated by a Java web service that was deployed in an application server.

We evaluated the performance of the generated web service by launching a set of queries related to 200 genetic diseases. Service times ranged between 3 and 12 minutes using a server based on Windows Vista Ultimate™ with 4 GB of RAM. These timings include both the plan generation and the retrieval of all structures belonging to all proteins involved in the target genetic disorder.

We believe that the generated web service facilitates the execution of the corresponding search flow, considering that manually retrieving the 3D structure associated to a single protein takes in average 2 minutes.

In the next section we compare the proposed method to other existing approaches to integrate public online biomedical databases.

4 Discussion

In recent years, different approaches have been proposed in the literature to address the integration of web-based biomedical databases. This includes information linkage [10], mediator/wrapper based methods [11], ontology-based mediation approaches [12, 13], and automated planning techniques [14]. We believe that planning techniques are particularly well suited to integrate public online databases since, i) they can process queries not supported by information linkage-based methods, and ii) unlike mediation-based approaches, they do not require establishing complicated mapping relationships between the schemas of the databases.

The BACIIS system [14], based on classical planning methods, relies on a custom ontology called BaO that includes all the relevant classes of objects in the domain together with a set of relationships that represent the inputs/outputs accepted/provided by the different databases. Queries are executed by generating a query processing plan—i.e. a sequence of searches on different databases—using the information provided by the ontology. Plans are automatically created using a modified version of the domain-independent GraphPlan [15] planning algorithm. The main drawback of this approach is that domain knowledge—i.e. the search flows—used by experts to manually process complex queries is not exploited to facilitate the creation of the plans.

Conversely, our method exploits the expert knowledge provided by the search flows to generate web search services that are similar to how human experts manually execute the queries. Besides, the DMs implemented by previously created web services can be reused as components supporting complex query services.

5 Conclusions

In this paper, we propose the use of HTN planning techniques to create complex web search services over multiple public online biomedical databases. HTN planning provides a framework to encode manual search flows used by biomedical researchers as HTN planning problems. Planners created to solve these HTN problems can be used as mediators that perform searches in a similar way as human experts execute queries.

Besides, the generated web services can be reused as components to build more complex query services, thus facilitating the integrated access to multiple public online biomedical resources.

References

1. Galperin, M.Y.: The Molecular Biology Database Collection: 2008 Update. Nucleic Acids Research 36(Database issue), D2–D4 (2008)
2. `http://www.ncbi.nlm.nih.gov/pubmed/` (last accessed, April 2008)
3. `http://www.ncbi.nlm.nih.gov/sites/entrez?db=omim` (last accessed, April 2008)
4. `http://www.eddnal.com/` (last accessed, April 2008)
5. Sacerdoti, E.: The nonlinear nature of plans. In: Proceedings of the International Joint Conference on Artificial Intelligence, pp. 206–214 (1975)
6. Erol, K., Hendler, J., Nau, D.: Semantics for hierarchical task-network planning. Technical Report CS TR-3239, UMIACS TR-94-31, ISR-TR-95-9. University of Maryland (1994)
7. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Practice. Morgan Kaufmann, San Francisco (2004)
8. Nau, D., Au, T.C., Ilghami, O., Kuter, U., Murdock, J.W., Wu, D., Yaman, F.: SHOP2: An HTN Planning System. Journal of Artificial Intelligence Research 20, 379–404 (2003)
9. `http://www.cs.umd.edu/projects/shop/description.html` (last accessed, April 2008)
10. Dias, G., Oliveira, J.L., Vicente, F., Martín-Sánchez, F.: Integrating Medical and Genomic Data: a Successful Example of Rare Diseases. Stud. Health Technol. Inform. 124, 125–130 (2006)
11. Haas, L.M., Schwarz, M., Kodali, P., Kotlar, E., Rice, J.E., Swopre, W.C.: DiscoveryLink: A system for Integrated Access to Life Sciences Data Sources. IBM Systems Journal 40(2), 489–511 (2001)
12. Stevens, R., Baker, P., Bechhofer, S., Ng, G., Jacoby, A., Paton, N.W., Goble, C.A., Brass, A.: TAMBIS: Transparent Access to Multiple Bioinformatics Information Sources. Bioinformatics 16(2), 184–186 (2000)
13. Alonso-Calvo, R., Maojo, V., Billhardt, H., Martín-Sánchez, F., García-Remesal, M., Pérez-Rey, D.: An agent- and ontology-based system for integrating public gene, protein and disease databases. Journal of Biomedical Informatics 40(1), 17–29 (2007)
14. Miled, Z.B., Li, N., Bukhres, O.: BACIIS: Biological and Chemical Information Integration System. Journal of Database Management 16(3), 72–85 (2005)
15. Blum, A., Furst, M.: Fast Planning Through Planning Graph Analysis. Artificial Intelligence 90, 281–300 (1997)
16. `http://www.orpha.net/consor/cgi-bin/index.php` (last accessed, April 2008)
17. `http://www.ncbi.nlm.nih.gov/sites/entrez?db=gene` (last accessed, April 2008)
18. `http://www.genenames.org/` (last accessed, April 2008)
19. `http://www.expasy.ch/sprot/` (last accessed, April 2008)
20. `http://www.rcsb.org/pdb/home/home.do` (last accessed, April 2008)
21. `http://www.ebi.ac.uk/msd/` (last accessed, April 2008)