Knowledge-driven Model Assembly and Execution

Alfredo Gabaldon and Natarajan C. Kumar

GE Research, Niskayuna NY

Abstract

We report on a system under development that uses knowledge graphs—knowledge bases that use graphs as the main data structure—of scientific knowledge to assemble scientific models in response to users' requests to compute the value of a set of "output" variables from a set of given "input" variables. A user may also upload observational data and request the system to assemble models as hypotheses that explain the observed data.

1 Introduction

Scientists with a dataset containing values for a set of variables $\vec{x}$ who want to compute the value of a variable y that depends on $\vec{x}$ may not immediately know how to compute y, and after figuring that out they still need to implement the programs that will do the calculations. It may be the case they can use previously developed code for parts of the calculation, provided they have a good understanding of the assumptions underlying the code. The scientists would also need to put together the different pieces of code, new or reused, into a complete model for the desired calculation. It is easy to imagine that, except for very simple cases, the process of creating such a model is time consuming and requires scientists to be good programmers in addition to their scientific expertise. Moreover, reusing existing models, especially written by others, can be challenging if the code is not accompanied by thorough documentation that explicitly and clearly describes under what assumptions the code can be applied. This is one of the challenges that has recently been put forward by several DARPA programs such as World Modelers and ASKE. We report on our work on a system that constructs scientific models on demand given a user's query asking for the value of some quantity given the values of a set of input quantities or in response to the user uploading observational data. The system uses a repository of scientific background knowledge in the form of a knowledge graph, both to understand the user's query and to assemble a model to compute the answer. As a user interface, the system uses controlled-English for query expression and to present some of the results back to the user. The controlled-English language is based on a language called SADL (Semantic Application Design Language) [1] developed at GE that uses semantic technologies to provide formal meaning to the language and to make it computer processable. Scientific knowledge is meant to be obtained by semi-automated extraction from code and scientific documents, but we do not address the extraction problem here. The extracted knowledge is assumed to have been stored in a knowledge graph that complies with W3C standard languages OWL, RDF and Sparql. Our system uses a model execution framework based on Dynamic Bayesian Networks (DBNs) which is capable of performing various inference tasks: prognostics, diagnosis, sensitivity, and others. Although this limits us to models that can be cast as DBNs, the system architecture does not preclude the use of a different execution framework as long as it is expressive enough to handle typical scientific variable relationships.

Significance We believe that systems like the one described in this paper will provide scientists with powerful tools for creating models faster and with less effort. It will also help them discover

models they may not have considered otherwise and make it easier to find explanations to surprising results. A critical requirement to achieving this is the use of knowledge graphs and semantic models to store knowledge about software modules and pieces of reusable code, together with relevant scientific background knowledge. This makes knowledge available for computation and reasoning in a very general way through standard query languages and inference procedures. Standard semantic languages ensure the content of the knowledge graphs is unambiguous, accessible by standard-compliant subsystems, and also shareable.

2 Example Domain and Scenario

We use an example to illustrate the components of the system we are developing. While the example involves a small set of equations, it is complex enough to raise various questions that need to be addressed by the system. In this project we are using hypersonic aerodynamics as a use-case domain, given its strategic importance to GE and government agencies. Additionally, NASA's Guide to Hypersonics provides an excellent source of code and documents to experiment with extraction. The particular example we use involves computations of air flow variables: Total Temperature and Total Pressure computed from Altitude and Air Speed. The diagram in Figure 1 represents how variables influence each other. Static Temperature and Static Pressure are determined by Altitude. Speed of Sound depends on the Static Temperature. Mach speed depends on the Speed of Sound and the Air Speed of the moving object. Total Temperature is determined by the Static Temperature and Mach speed. Similarly for Total Pressure.

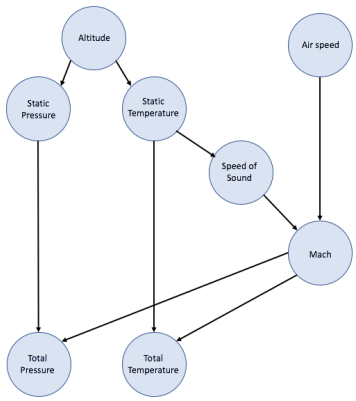

Figure 1: Air flow variable influence diagram

3 Knowledge Repository

We assume the scientific knowledge, whether extracted from code, documents, and/or manually curated, is stored as knowledge graphs. These are representations of knowledge based on the concept of a graph and are typically stored in repositories that use node-property-node triples as the main conceptual structure, unlike tables in traditional databases. Moreover, we assume knowledge graphs that include semantic models (aka ontologies) that describe and annotate the content using standards such as OWL, RDF and Sparql.[1]. Since knowledge extraction is the goal of separate, concurrent efforts, to support our work we use hand-crafted models of our example scenario that are close to what we expect can be realistically extracted. We created a domain model for the air flow variables shown in Figure 1 that includes specifications of equations for each variable

[1] `https://www.w3.org/standards/`

in the diagram, excluding Altitude and Air speed which are inputs in the example. In general, the relationship between a set of variables may be defined by an explicit equation or by a function that is only defined by a piece of code, what we refer to as an "external equation." In addition to an equation, each node (variable) is defined with additional information such as a value range and a default distribution. This is useful for computations where the variable cannot be computed from the given inputs but can be treated as a random variable.

Given a user's request to compute the value of some variable given some inputs, the system will access the knowledge graph to infer a *computational graph*: a subset of equations that minimally capture the network of dependencies from the desired output variables to the given input variables. The computational graph will then be translated into the required input language of a model execution framework, in our case, a DBN execution system previously developed at GE [2].

3.1 Knowledge Graph

The information stored in the knowledge graph can be divided into three groups: domain knowledge, domain-independent knowledge, and meta-level knowledge. Domain knowledge is the content from knowledge extraction and curation. Domain-independent knowledge includes semantic models that are used to represent extracted knowledge but are independent of any particular subject matter and so can be reused across domains. Meta-level knowledge represents information about activity that occurs in the system, including queries that have been submitted by a user, scientific models that have been constructed in response to a query, results obtained from executing a model, and so on.

The domain-independent model includes definitions of entities like "UnittedQuantity," and "Equation". The semantic model for this part of the knowledge graph is hand-crafted in SADL and maintained as part of the overall system. A united quantity is essentially a variable in a scientific model, for example, the speed of sound, and has values and units as properties. An equation specifies inputs (or arguments), outputs (return types), and an expression to be used for the actual computation, or in the case of an external equation, e.g., a piece of code, a link to a location where the code is stored and its identifier (a URI). Another concept defined here is the notion of a "Computational Graph" and a subtype for representing DBNs. The DBN concept in the semantic model links to a node (a variable), an equation or an external equation, a type (deterministic, stochastic, discrete, continuous, etc), a distribution (the DBN execution framework currently supports uniform, normal, Weibull, lognormal, exponential, beta, binomial, poisson and a few others), and a range of values. Other than node and equation, these properties are treated as defaults and may be ignored or overridden by the user. The domain-independent model encoded in SADL includes the statements in Figure 2.

```
^Equation is a class
        described by input with values of type UnittedQuantity
        described by output with values of type UnittedQuantity
        described by expression with values of type string
        described by assumption with values of type Condition.

^ExternalEquation is a type of ^Equation
        described by externalURI with values of type anyURI
        described by location with values of type string.


DBN is a type of ComputationalGraph
        described by node with values of type UnittedQuantity
        described by hasEquation with a single value of type ^Equation
        described by hasModel with a single value of type ^ExternalEquation
        described by ^type with values of type NodeType
        described by distribution with a single value of type Distribution
        described by range with values of type Range.
```

Figure 2: Domain-independent model definitions.

```
StaticTempEq is a type of ^Equation.
input of StaticTempEq has at most one value of type Altitude.
output of StaticTempEq only has values of type StaticTemperature.
expression of StaticTempEq always has value "518.6-3.56 * Altitude /1000.0".

StaticTempEq2 is a type of ^Equation.
input of StaticTempEq2 has at most one value of type Altitude.
output of StaticTempEq2 only has values of type StaticTemperature.
expression of StaticTempEq2 always has value "389.98".

StaticTempDBN is a type of DBN.
hasEquation of StaticTempDBN has at most one value of type {StaticTempEq or StaticTempEq2}.
range of StaticTempDBN always has value (a Range with lower -200 with upper 200).
distribution of StaticTempDBN always has value uniform.
```

Figure 3: Definitions for Static Temperature.

As mentioned earlier, we are using hand-crafted representations of knowledge for the domain-specific part. The knowledge graph includes definitions of the variables shown in Figure 1 and corresponding equations/DBNs. The definitions for Static Temperature are shown in Figure 3. Static temperature has multiple equations that apply for different values of Altitude. In general, equations will be associated with assumptions under which they are applicable and the system will need to check these assumptions to determine if a computational graph using a particular equation can be chained together with another.

Finally, our knowledge graphs use a semantic meta-model for persisting queries, computational graphs, and execution events. The meta-model defines a class CCG for "complex" computational graphs comprised of a set of subgraphs each linked to the output variable with its computed value. CCG instances store the structure of the assembled model and the computed value of each node in the computational graph. A concept called CGExecution is used to represent execution events and its definition includes start and end times, a link to the computational graph used (a CCG), and a measurement of the accuracy of the results obtained. The concept CGQuery is used to capture user queries. A CGQuery links to the inputs given in the query, the output, and an CGExecution instance. The query output should be the same as the output of the leaf subgraph in the computational graph. We intend to keep the meta-model definitions general enough to accommodate computational graphs that are not based on DBNs.

3.2 SADL Dialog Interface

The user interface for the system is a controlled-English dialog editor that lets users enter queries and get answers interactively in a "chat" style dialog. SADL implements a controlled-English grammar with the expressivity of OWL1 plus qualified cardinality constraints from OWL2, as well as rules. The SADL Dialog Editor interface is driven by a specified knowledge graph that serves as the domain of discourse. It uses this knowledge graph to recognize domain concepts used in queries by the users and to process them accordingly. The dialog interface can be used for general domain knowledge queries, such as "What is StaticTemperature?" In our project, the focus is on queries that result in scientific model assembly and execution to compute an answer, as well as on elaboration and follow-up queries. In our example scenario, the user might ask:

What is the ^value of TotalTemperature and the ^value of TotalPressure when the ^value of Altitude is 30000 and the ^value of AirSpeed is 1000?

SADL uses syntax highlighting to indicate recognized entities from the background knowledge. Given such a query, the system tries to assemble a model to compute Total Temperature and Pressure from the given input values. We look at how this process works in the next section.

4 Model Assembly and Execution

When a user submits a query, the system accesses the knowledge graph looking for a set of equations that can be linked together to compute an answer from the given inputs. Consider a simplified version of the above query: What is the ^value of TotalTemperature when the ^value of Altitude is 30000? Having recognized the concepts, the system attempts to put together a set of equations to compute Total Temperature from Altitude. The system first infers a variable dependency graph and adds that information to the knowledge graph. In our example, the dependency graph corresponds to the graph depicted in Figure 1. The system will then find all the variables in the knowledge graph that Total Temperature depends on, shown in yellow and green in Figure 4. Altitude and Air speed are root nodes, but only Altitude is a given as an input in the query. Yellow nodes are variables whose equations may be needed for the model. In a second step, the system looks at

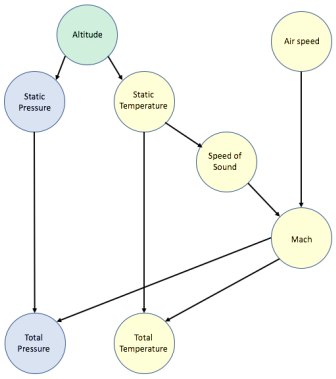

Figure 4: Ancestors of Total Temperature.

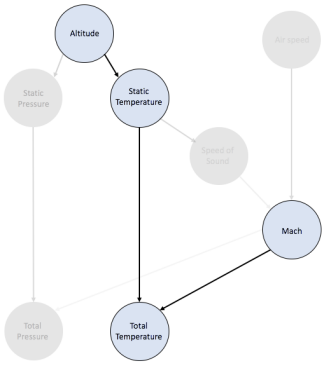

Figure 5: Final graph for Total Temperature.

descendants of the inputs given in the query and eliminates nodes that depend on variables that are not among the given inputs. In the example, this will remove Air speed and Mach. Total Temperature depends on Mach, so removing the Mach equation from the set means Mach will be treated as an input. Air speed, on the other hand, is just removed from the subnetwork because it is completely independent of the given inputs. In Bayesian terms, all other nodes are independent of Air speed given Mach. Since Mach is not a given input in the query, i.e., we don't have a value for it, the DBN execution framework treats it as a random variable and employs sampling using the specified distribution for Mach. One of the advantages of the knowledge graph is that it allows very complex graph pattern-matching, which lets us build this computational subgraph using a single Sparql query (once the dependency graph has been inferred). Finally, equations whose output is not required by any other equation are removed from the set. In the example this step removes the Speed of Sound equation. The resulting model for answering the query is shown in Figure 5. Once the system builds a computational graph for the desired output, it has to be translated into the format required for execution. For our DBN Execution Framework, the system takes the computational graph information and creates a DBN specification in json format then launches the model execution through a REST service provided by the DBN Execution Framework. For the example query above, the execution framework will use the established random distribution assigned to the Mach variable, and compute a probability distribution (or histogram in a discrete sense) of the total temperature. The DBN framework can also effectively address other query scenarios: *calibration*–update parameters in the model when data for outputs is provided; *sensitivity*–compute the amount of variance contribution to the output from the uncertainty of each of the different inputs; and *optimization*–run an optimization to compute input settings or distributions that result in a max/min output value.

The system needs to address situations where only a partial model is found, or the model found doesn't use all the inputs, etc. We discuss how the system will handle some of these cases in the context of hypotheses in the next section.

5 Meta-Reasoning and Hypothesis Generation

One of the processing tasks that is triggered by a query is the ingestion of the query itself into the knowledge graph. When we persist a query, we essentially create an instance of the concept CGQuery. This instance will be linked to scientific concepts used as inputs, e.g., Altitude, and the provided values. After model execution, the query instance will also be linked to an instance of a CGExecution instance. The CGQuery instance for the example, in SADL, is as in Fig. 6. Each

```
cgq1 is a CGQuery
       input (an Altitude with ^value 30000)
       output (a TotalTemperature)
       with execution (a CGExecution cge1 with compGraph cg1).
```

```
cg1 is a CDBN
   subgraph sg1
   subgraph sg2
   subgraph sg3.

sg1 is a SubGraph
       cgraph (a StaticTempDBN hasEquation (a StaticTempEq))
       output (a StaticTemperature ^value 483).

sg2 is a SubGraph
       cgraph (a MachSpeedDBN distribution uniform)
       output (a MachSpeed ^value 1.0 ^value 1.5 ^value 2.0).

sg3 is a SubGraph
       cgraph (a TotalTemperatureDBN hasEquation (a TotalTemperatureEq))
       output (a TotalTemperature ^value 2000 ^value 2500 ^value 3000).
```

Figure 6: Query (left) and computational graph (right) instances, expressed in SADL.

subgraph is linked to a DBN and either an equation or a distribution. Since Mach is sampled, the corresponding subgraph sg2 links to a MachSpeedDBN instance with a distribution. The distribution is persisted because the user may have overridden the default distribution. As for values, Static temperature is a deterministic node and has a single computed value. Mach speed was sampled, so it has multiple values, and so does Total temperature, which takes Mach as one of its inputs. All this information is available in the knowledge graph after the execution of the model. The user is able to look not only at the output value but also at the values of all intermediate nodes. The subgraph elements are also available for inspection. The user can inspect intermediate nodes and submit follow-up questions about the scientific concepts used in the model.

All this information in the knowledge graph allows the system to perform meta-reasoning [3], i.e., to reason over current and previous queries, models, and results. This is useful is to explain results to the user. Consider for example the query:

What is the ^value of MachSpeed when the ^value of Altitude is 30000 and the ^value of AirSpeed is 2000?

The user gets an answer of 2.947, then tries with an altitude of 40000 and gets an answer of 3.028. Then she tries an altitude of 50000 and gets the same answer of 3.028. The user may wonder why she got the same result. Using the meta-data, the system can tell the user that for the first query Static temperature was computed using equation 518.6 - 3.56 * Altitude / 1000, whereas for the second and third queries Static temperature is a constant 389.98. Furthermore, using background knowledge the system may explain that the temperature in the lower stratosphere (between altitudes of 36152 and 82345 ft) is constant. We have implemented ingestion of query, model, and answer meta-data, but have only taken first steps toward providing explanations. For example, the user can ask what equations were used in any assembled models that were used to answer a query. The system also displays models graphically. In future work we intend to extend SADL dialog to provide more detailed and useful explanations. We will also consider using meta-reasoning in the model assembly process, which could be beneficial in scenarios with large models.

In addition to queries, the user may have observational data and wish to find a model that best explains the data. In this case we would like the system to automatically hypothesize models that explain the data. The system would process the request in different ways depending on the variables present in the data and the existing models in the knowledge graph:

1. The system finds at least one model—the system succeeds collecting submodels to assemble one or multiple models that predict the observed variable. There are multiple scenarios:
   (a) All inputs needed are available in the observed data, and all assumptions are satisfied. The system finds and/or assembles several models that include all variables present in the data. It then executes the models to provide the user the following information:

(i) A measure of each model's accuracy (e.g., mean error) (ii) A sensitivity analysis to determine which input is the main contributor to the output (iii) If the error measure is non-random or if there is a bias in the error, then the system will attempt to compute a delta function (a model of the error), which can be a regression model, a neural network, or some other type of model. Then the new model with the delta term is a hypothesis and is added to the knowledge graph.
The most accurate models are then presented to the user in a readable form. The results will also include the assumptions made by each model and the accuracy.

(b) Not all inputs needed by the model are available, and all assumptions are satisfied.
In this case, the system will find at least one model that can compute the desired output. For each model, the system will automatically sample the missing inputs based on the variable's probability distribution, and compute the likelihood of the model predicting the observation. The models with highest likelihood are then presented to the user with additional information (such as sensitivity analysis) as in the previous item.

(c) All inputs are available, but some assumptions are not satisfied.
The system finds a candidate model that includes all the observed variables whose assumptions are not satisfied. In this case the system can present to the user the submodel and the corresponding assumptions that are not satisfied by the data and also provide that feedback to knowledge extraction/curation systems to look for alternatives, e.g., equations or submodels that apply under different assumptions.

2. The system fails to assemble a model that may explain the data.
This can be due to several reasons, the most prominent of which is when an input variable in the observed data is not even recognized. The system will then compute measures of correlation between the input and the output from the observed data to determine if any input to output relationship exists that are not captured by the knowledge graph. If there are, then this information can be both presented to the user and passed on as feedback to knowledge extraction/curation systems which can then search for and extract the missing knowledge. It is also possible that the user knows of the missing knowledge, for example, of an equation that relates some variables, and they may enter the knowledge on the dialog interface. If the system fails to assemble the model for any other reason, it will be communicated clearly to the user, so that the user can then identify additional issues in constructing the model for the collected observed data.

Much work remains. We have only implemented the first case above. Implementing the other cases is in our future work plans as well as removing a number of simplifying assumptions.

6 Acknowledgments

This work was supported by DARPA under Agreement No. HR00111990007.

References

[1] "Toward a Unified English-like Representation of Semantic Models, Data, and Graph Patterns for Subject Matter Experts", A Crapo and A. Moitra, *Semantic Computing*, 2013.

[2] "Dynamic Bayesian network for aircraft wing health monitoring digital twin," C. Lia, S. Mahadevana, Y. Ling, S. Chozeb, and L. Wang. *AIAA Journal*, 2017.

[3] "Metareasoning: A Manifesto", M.T. Cox and A. Raja. *AAAI*, 2008.