The Hetero-functional Graph Theory Toolbox

Prabhat Hegde\textsuperscript{1,*}, Wester C. H. Schoonenberg\textsuperscript{1}, Dakota Thompson\textsuperscript{1}, and Amro M. Farid\textsuperscript{1}

\textsuperscript{1}Dartmouth College, Thayer School of Engineering, Hanover, 03755, USA
\textsuperscript{*}Prabhat.Hegde.TH@dartmouth.edu

ABSTRACT

In the 20\textsuperscript{th} century, newly invented technical artifacts products were connected to form large-scale complex engineering systems. Furthermore, the interactions found within these networked systems has grown in both degree as well as heterogeneity. Consequently, these already complex engineering systems have converged in what is now called systems-of-systems. The analysis, design, planning, and operation of these engineering systems from a holistic perspective has necessitated ever-more sophisticated modeling techniques. Despite significant advancements in model-based systems engineering and network science, these seemingly disparate fields have experienced similar limitations in addressing the complexity of engineering systems. Hetero-Functional Graph Theory (HFGT) has emerged as a means to address some of these limitations. This paper serves as a user guide to a recently developed Hetero-functional Graph Theory Toolbox which facilitates the computation of HFGT mathematical models. It is written in the MATLAB language and has been tested with v9.6 (R2019a). It is openly available on GitHub together with a sample input file for straightforward re-use. The paper details the syntax and semantics of the input file, the principal data structure of the toolbox, and the functions used to construct and populate this data structure. The toolbox has been fully validated against several peer-review HFGT publications.

Introduction

In the 20\textsuperscript{th} century, newly invented technical artifacts products were connected to form large-scale complex engineering systems. Indeed, the electric generator, the telephone, the petro-chemical refinery, the automobile and a whole host of medical treatment and imaging devices have given rise to the electric power, communication, oil & gas pipeline, transportation, and healthcare delivery systems that we know today\textsuperscript{1}. Over time, these engineering systems have evolved to incorporate newer technologies that rely on multiple engineering systems (e.g. renewable energy, mobile phones, fuel-cells, electric-vehicles, and wearable-health technologies). Consequently, the interactions found within these networked systems has grown in both degree as well as heterogeneity. Furthermore, these already complex engineering systems have converged in what is now called systems-of-systems. The “smart” grid, the energy-water nexus, electrified transportation systems, the energy-water-food nexus, and the development of interdependent smart-city infrastructure are all examples of how contemporary systems-of-systems are becoming an integral part of human life\textsuperscript{2}. This paper regards such systems as engineering systems:

Definition 1. Engineering System\textsuperscript{1} : A class of systems characterized by a high degree of technical complexity, social intricate, and elaborate processes aimed at fulfilling important functions in society.

The analysis, design, planning, and operation of these engineering systems from a holistic perspective has necessitated ever-more sophisticated modeling techniques. Two Informatics science are of particular relevance. Model-Based Systems Engineering (MBSE) is a practical and interdisciplinary engineering field that enables a successful realization of complex systems from concept, through design, to full implementation\textsuperscript{3}. Often, the practice of MBSE relies on the use of UML\textsuperscript{4} and/or SysML\textsuperscript{5,6} which consists of several graphical viewpoints of the function and form of the engineering system\textsuperscript{7}. These viewpoints include block diagrams, activity diagrams, and state-machine diagrams; making them well-equipped to describe the complex and heterogeneous nature of these systems; both graphically and later in simulation. In the meantime, network science has matured as a discipline to provide quantitative analyses for the structure and function of networks that appear across natural, social and engineering sciences\textsuperscript{8,9}. The spatially-distributed nature of engineering systems has often times led to their underlying models being rooted in graph theory. Thus, it has been applied to systems like transportation systems\textsuperscript{10}, power grids\textsuperscript{11,12}, water networks\textsuperscript{13}, supply chains\textsuperscript{14}, and healthcare systems\textsuperscript{15}.

Despite significant advancements, these seemingly disparate fields have experienced similar limitations in addressing the inherent complexity of engineering systems. While the graphical models used in MBSE often serve as the basis for developing complex simulations of system behavior, they fall short in providing a quantitative analysis of system structure. On the other hand, network science’s reliance on graphs as a data structure limits its ability to handle the explicit heterogeneity one often encounters in engineering systems. Even the recent developments toward multi-layer networks have been shown to exhibit
several modeling constraints that inhibit the representation of an arbitrary number of network layers of arbitrary topology connected arbitrarily\textsuperscript{16,17}. The trend toward greater and more heterogeneous interaction between multiple engineering systems is only set to accelerate given the continual proliferation of diverse physical and information technologies\textsuperscript{18–22}. The current methodological and theoretical limitations in the MBSE and network science fields necessitates new mathematical modeling techniques for multiple integrated engineering systems.

Hetero-Functional Graph Theory (HFGT) has emerged as a means to model the structure and function of highly interconnected and heterogeneous engineering systems\textsuperscript{17}. In order to support insightful quantitative analyses, HFGT relies on multiple graphs as data structures. It also explicitly incorporates the heterogeneity of conceptual and ontological constructs found in MBSE. In doing so, it facilitates the translation of systems engineering models (e.g. SysML) to a mathematical engineering systems description, providing a rigorous platform for modeling systems-of-systems. The foundational HFGT works are in the field of mass-customized production systems\textsuperscript{23–26}. In some ways, such production systems present modelling challenges that are common to most engineering systems. The production capabilities of a production system can come together in various permutations and combinations to produce an almost infinite number of product variants. At the same time, their structure and behavior is dynamically changing. Since these first works, the theory has methodologically evolved to provide qualitative and quantitative analyses in other application domains including electric power systems\textsuperscript{27,28}, energy-water nexus\textsuperscript{17,29–31}, electrified transportation systems\textsuperscript{32–34}, microgrid-enabled production systems\textsuperscript{35}, industrial energy management\textsuperscript{36}, personalized healthcare delivery systems\textsuperscript{37,38} and interdependent smart city infrastructures\textsuperscript{17}.

Hetero-functional graph theory is composed of six mathematical models that together form a System Adjacency Matrix $A_S$ as its seventh.

1. System Concept $A_S$
2. Hetero-functional Adjacency Matrix $A_{\rho}$
3. Controller Agency Matrix $A_Q$
4. Controller Adjacency Matrix $A_C$
5. Service as Operand Behavior $N_L$
6. Service Feasibility Matrix $\Lambda$

This paper serves as a user guide to a recently developed \textit{Hetero-functional Graph Theory Toolbox}\textsuperscript{39} which facilitates the instantiation of these seven mathematical models from a single XML input file. It is written in the MATLAB\textsuperscript{40} language and has been tested with v9.6 (R2019a). It is openly available on GitHub\textsuperscript{39} together with a sample input XML file for straightforward re-use.

The remainder of the paper proceeds as follows. The section entitled: “Creating a HFGT Input File” details how to translate a real-world system into an HFGT Toolbox input XML file. The following section entitled “HFGT Data Structures and Functions” describes the \texttt{myLFES} (LFES: Large Flexible Engineering System) data structure which organizes all of the data generated by the toolbox. This section also describes the functions that transform the input XML file into an empty version of the \texttt{myLFES} data structure. Lastly, it describes the functions that populate the \texttt{myLFES} data structure with HFGT values. The section entitled “Toolbox validation” describes how the HFGT toolbox has been validated against previously published results. Finally, the “Conclusions” section brings the paper to a close. This paper presumes that the reader has a working knowledge of HFGT which is otherwise gained from a thorough reading of the associated text\textsuperscript{17}. Furthermore, the HFGT toolbox uses an XML input file and so a basic knowledge of the eXtensible Markup Language\textsuperscript{41} is needed. Finally, UML/SysML models\textsuperscript{5} are used to convey the object-oriented programming data structures found throughout the HFGT toolbox.

### Creating a HFGT XML Input File

As a high-level overview, the purpose of the HFGT input file is to provide a structured representation of the data associated with an instantiated large flexible engineering systems. To that end, Figure 1 shows the meta-architecture of the system form of a Large Flexible Engineering System (LFES) as described by HFGT\textsuperscript{17}. HFGT assumes that the formal elements of any LFES can be viewed as instances of this meta-architecture. The input data file, thus, organizes the resources present in an engineering system into these “meta-elements” and captures the functions (i.e. methods) that they are capable of performing. Such a complex hierarchical structure is readily stated in the eXtensible Markup Language (XML)\textsuperscript{41}. Furthermore, XML files are both human and machine readable; making it easy to catch any potential errors that are inadvertently introduced into the input file.
The first step to using the HFGT toolbox is to accurately write an input XML file that instantiates the meta-architecture shown in Figure 1. Figure 2 provides an example of such an XML file. Each line of the XML file is composed of a start or end XML element. The root of the input XML file is the LFES element. All of the meta-elements shown in Figure 1 are now explained in turn.

**LFES**: This element is the root of the input XML file and indicates an instantiated LFES. All of the meta-elements shown in Figure 1, and the processes that they are capable of carrying out are contained within this root. The attributes “name” and “type” describe the name of the system and its type. The attribute “dataState” is defaulted to “raw” to indicate its pre-processed state.

**Machine**: This XML element is representative of a “Transformation Resource” (M) shown in Figure 1. The “name” attribute describes the name of the transformation resource. If a controller has agency over the transformation resource, the former’s name is stored under the “controller” attribute. The attributes “gpsX” and “gpsY” capture the physical location of the transformation resource with respect to a reference axes. The illustrative example in Figure 2 shows a water treatment facility as an instantiation.

### Figure 2. An example HFGT Toolbox input XML input file

```xml
<lfes version="1.0" encoding="UTF-8">
  <lfes name="Tranformica-Hump" type="Smart City" dataState="raw">
    <operations>
      <transport>
        <controller name="Water Utility">transport</controller>
        <methodPair name="water" preset="potable water" methodLinkName="treat water" methodLinkRef="potable water"/>
        <methodPair name="cold water" preset="potable water" methodLinkName="treat water" methodLinkRef="potable water"/>
        <methodPair name="hot water" preset="potable water" methodLinkName="treat water" methodLinkRef="potable water"/>
      </transport>
      <transport>
        <controller name="Water Utility">transport</controller>
        <methodPair name="water" preset="potable water" methodLinkName="treat water" methodLinkRef="potable water"/>
        <methodPair name="cold water" preset="potable water" methodLinkName="treat water" methodLinkRef="potable water"/>
        <methodPair name="hot water" preset="potable water" methodLinkName="treat water" methodLinkRef="potable water"/>
      </transport>
      <transport>
        <controller name="Water Utility">transport</controller>
        <methodPair name="water" preset="potable water" methodLinkName="treat water" methodLinkRef="potable water"/>
        <methodPair name="cold water" preset="potable water" methodLinkName="treat water" methodLinkRef="potable water"/>
        <methodPair name="hot water" preset="potable water" methodLinkName="treat water" methodLinkRef="potable water"/>
      </transport>
    </operations>
  </lfes>
</lfes>
```
of a transformation resource.

**MethodxForm**: This XML element is used to describe the ability of a transformation resource to transform an operand. As shown in Figure 1 and as reflected in the illustrative example in Figure 2, this element is nested within the Machine (i.e. transformation resources) element. The attributes “name”, “status”, “operand”, and “output” describe the name of the process, its status (active/inactive), the set of operands needed for the process, and the set of outputs as a result of the transformation respectively.

**MethodxPort**: As shown in Figure 1, the operations involving holding or transportation of an operand are universal to all of the physical resources. Rather than introduce an XML element for holding and transportation processes separately, the XML file format introduces this element to refer to a “refined transportation process” as their combination. Furthermore, a storage process is considered a transportation with the same origin and destination. The attributes used to describe these processes include “name”, “status”, “origin”, and “dest”, which describe the name, the active state, the origin and the destination of the refined transportation process respectively. The attribute “ref” describes the refinement of the transportation in terms of a string that reflects the associated holding process. The operands needed to carry out the process and the output of the process are described by the attributes “operand” and “output” respectively.

**IndBuffer**: This XML element is representative of an “Independent Buffer” (B) as shown in Figure 1. The “name” attribute describes the name of the independent buffer. If a controller has agency over the independent buffer, the former’s name is stored under the “controller” attribute. The attributes “gpsX” and “gpsY” capture the physical location of the independent buffer with respect to a reference axes.

**Transporter**: This XML element represents a “Transportation Resource” (H) as shown in Figure 1. The attributes “name” and “controller” captures its name and the controller that controls the resource respectively.

**Controller**: As shown in Figure 1, controllers have jurisdiction or agency over physical resources. The association of a physical resource to an independent controller is captured in the attributes of the physical resource itself. Dependent controllers embedded within a physical resource are implicit to the tool box’s functionality and are not explicitly stated in the input XML File. Thus, the “controller” XML element describes independent controllers exclusively. The attribute “name” describes the name of the controller and the attribute “status” describes whether the controller is active or inactive.

**PeerRecipient**: This XML element describes the peer controllers that receive information from the controller named in the controller XML element. The name attribute provides the name of this peer controller.

**Service**: The services shown in Figure 1 describe the evolution in the state of an operand as a delivered service modeled as a Petri Net. The attribute “name” describes the name of the service and the attribute “status” describes whether the service is active or inactive in the system.

**ServicePlace**: This XML element describes the places of the service (Petri) net. Each place is given its own name under the “name” attribute.

**ServiceTransition**: This XML element describes the transitions of the service (Petri) net. Each transition is given its own name under “name” attribute. The “preset” and “postset” attributes refer to the respective names of the places that send tokens to or receive tokens from the service transition named in this XML element. The “methodLinkName” attribute indicates the name of the system process to which the transition is linked. The “methodLinkRef” attribute indicates the refinement of the transportation process to which the transition is linked (if any). As shown in Figure 1, services and physical resources are associated by virtue of the link between the service transitions in the former and the system processes provided by the latter.

**Abstractions**: This XML element has no attributes associated with it. It contains all possible holding refinements and functional sequences (i.e. MethodPairs) in the system nested within it.

**MethodxPort** (within the Abstractions Element): This XML element contains information about all possible refinements in the system. In the illustrative example in Figure 2, the “refinements” possible while transporting the Electric Vehicle (EV) are: parking the EV, charging it by wire, discharging it and charging it wirelessly.

**MethodPair** (within the Abstractions Element): This XML element captures the possibility of one system process following another in the system. In Figure 2, the possibility that water being transported from an origin to a destination can be followed by a transportation from the new origin to a new destination has been described.
HFGT Data Structures and Functions

The Hetero-functional Graph Theory Toolbox uses Matlab’s object-oriented programming functionality to enhance modularity, extensibility and reusability. The primary purpose of the HFGT toolbox is to instantiate and subsequently populate the data in the myLFES structure. It’s associated class diagram is shown in Figure 3. The myLFES stores all of the information in the XML input file and then calculates all of the mathematical quantities identified in HFGT. This high-level purpose is achieved through two principal modules: XML2LFES() and raw2FullLFES() that are executed in sequence as shown in Figure 4. In brief, the XML2LFES() module serves to import the input XML file and create the myLFES data structure in a “raw” structure. Then, the raw2Full() module makes the HFGT calculations necessary to convert the myLFES data structure to the “full” state. Each of these modules is now discussed in detail.
As previously mentioned, the XML2LFES() module serves to import the input XML file and create the myLFES data structure in a “raw” structure. This functionality is achieved through the sequence of functions shown in the activity diagram in Figure 5. Each of these functions are now explained in turn.

### XML2LFES()

1. **S = xml2struct(xmlfile)**: The xml2struct function (v1.8.0.0) is a freely available third-party MATLAB function that can be otherwise downloaded from the MATLAB Central website[^43]. In the context of the HFGT toolbox, this function reads the input XML file and converts it into an intermediate object S which replicates the hierarchy of the input XML file. The S structure is identical to the myLFES structure shown in Figure 3 with two exceptions:

   1. It does not contain the root attributes of myLFES. These are calculated later.
   2. At the lowest level of decomposition, S has a cell array of objects (e.g. S.myLFES.machines{:}.methodsxForm).

Because the xml2struct() function is relatively rigid, and there is not much control on its output S, the remaining functions in XML2LFES() serve to “reshape” S into the more desirable form myLFES without any loss of information.

2. **myLFES = initializeLFES()**: This function is the LFES constructor and consequently instantiates myLFES. The high-level integer attributes of myLFES are initialized to zero and the sub-classes are initialized as empty structured arrays. It also calculates the number of transformation and structural degrees of freedom in myLFES.DOFM and myLFES.DOFH respectively.

3. **myLFES = setupMachines(myLFES, S)**: This function assigns information to the sub-object myLFES.machines by “unpacking” S.myLFES.machines. It also adds the nameref and statusref attributes to the myLFES.machines.methodsxPort class. These refer to the name of the associated transportation process and its initial status respectively. This function also collates the set of all transformation processes in myLFES.setTransformProcess.

4. **myLFES = setupIndBuffers(myLFES, S)**: This function assigns information to the sub-object myLFES.indBuffers by “unpacking” S.myLFES.indBuffers. It also adds the nameref and statusref attributes to the myLFES.indBuffers.methodsxPort class. These refer to the name of the associated transportation process and its initial status respectively. It also increments the value of myLFES.DOFH with the capabilities of the independent buffers.

5. **myLFES = setupTransporters(myLFES, S)**: This function assigns information to the sub-object myLFES.transporters by “unpacking” S.myLFES.transporters. It also adds the nameref and statusref attributes to the myLFES.transporters.methodsxPort class. These refer to the name of the associated transportation process and its initial status respectively. It also increments the value of myLFES.DOFH with the capabilities of the transportation resources.

6. **myLFES = setupControllers(myLFES, S)**: This function assigns information to the sub-object myLFES.controllers by “unpacking” S.myLFES.controllers.
myLFES = setupServices(myLFES, S): This function assigns information to the sub-object myLFES.services by “unpacking” S.myLFES.services.

setupAbstractTransportationMethods(myLFES, S): This function assigns information to the sub-object myLFES.abstract by “unpacking” S.myLFES.abstract.

The following support functions are used within the functions that set up the elements of the system:

objB=getResourceAttributes(objA): This function copies the attributes of a resource from the intermediate structure S to the corresponding resource class object within the myLFES object. This is helpful in specifying the attributes of each type of meta-element within myLFES.

[objB, setProcess, DOF]=getResourceMethods(objA,setProcess,DOF,opt): This function copies information associated with a resource from the intermediate structure S to the corresponding resource class object within the myLFES object. The opt argument of the function follows the classification in Figure 2. It is assigned “M”, “B” or “H” for machines, independent buffers and transporters respectively. In addition, it computes the transportation and transportation capabilities.

objA=insertObjB(objA,objB,idxA): This function inserts the fields of objB into the equivalent fields of objA. This function is used as a support function within both getResourceAttributes and getResourceMethods to extract fields from structure S and fit it into the correct hierarchical location within myLFES.

myLFES=raw2FullLFES(myLFES)

As mentioned previously, the raw2FullLFES() module makes the HFGT calculations necessary to convert the myLFES data structure from the “raw” to the “full” state. This functionality is achieved through the sequence of functions shown in the activity diagram in Figure 6. Each of these functions are now explained in turn.

**Figure 6.** Activity diagram of the raw2Full module

myLFES=calcResourceIndices(myLFES): This function assigns the new attributes idxMachine, idxBuffer and idxTransporter to the sub-objects myLFES.machines, myLFES.indBuffers, and myLFES.transporters respectively. These attributes represent the numeric index of the resource for each type of physical resource in the system. In addition, it also assigns an attribute idxResource to the sub-objects myLFES.machines, myLFES.indBuffers, and myLFES.transporters which indicates the index of the resource in the set of resources.

myLFES=calcResourceCounts(myLFES): This function calculates the number of each type of resource present in the system. In doing so, it populates the following attributes: myLFES.numMachines, myLFES.numIndBuffers, myLFES.numBuffers, myLFES.numTransporters, myLFES.numResources, myLFES.numControllers and myLFES.numServices.
myLFES=calcResourceSet(myLFES): This function creates a new class object myLFES.resources with two attributes: names and idx. These are the set unions of the names and idxResource attributes in myLFES.machines, myLFES.indBuffers and myLFES.transporters.

myLFES=packSetTransformProcess(myLFES): This function computes the set of unique transformation processes and assigns it to myLFES.setTransformProcess. It also assigns the number of unique transformation processes to myLFES.numTransformProcess.

myLFES=makeSetTransportProcess(myLFES): This function computes the set of all possible transportation processes system and assigns it to myLFES.setTransportProcess. It also assigns the number of transportation processes to myLFES.numTransportProcess.

myLFES=makeSetTransportRefProcess(myLFES): This function computes the set of all possible refined transportation processes and assigns it to myLFES.setRefTransportProcess. It also computes the numbers of refined transportation processes and holding processes and assigns them to myLFES.numTransportProcess and myLFES.numHoldingProcess respectively.

myLFES=initializeKnowledgeBasesConstraintsMatrices(myLFES): This function creates appropriately sized but empty sparse matrices to the three knowledge bases (i.e. myLFES.JM, myLFES.JH, myLFES.JHref), the three constraint matrices (i.e. myLFES.KM, myLFES.KH, myLFES.KHref), and the three system concept matrices (i.e. myLFES.AM, myLFES.AH, myLFES.AHref).

myLFES=calcJM_KM_AM(myLFES): This function computes the transformation knowledge base (myLFES.JM), the transformation constraint matrix (myLFES.KM) and the resultant transformation system concept (myLFES.AM).

myLFES=calcMachineIdxPort(myLFES): This function computes the following five indices associated with a refined transportation process conducted by a given machine.

- myLFES.machines.methodsxPort.idxOrigin: the index of the resource where the transportation/holding process originates from the list of indices of all resources present in the system.
- myLFES.machines.methodsxPort.idxDest: the index of the resource where the transportation/holding process terminates from the list of indices of all resources present in the system.
- myLFES.machines.methodsxPort.idxHold: the index of the holding process from the list of holding processes in the system.
- myLFES.machines.methodsxPort.idxPort: the index of the transportation process from the set of transportation processes possible in the system.
- myLFES.machines.methodsxPort.idxPortRef: the index of the refined transportation process from the set of refined transportation processes possible in the system.

myLFES=calcIndBufferIdxPort(myLFES): This function is analogous to calcMachineIdxPort() above, but instead calculates the index attributes in myLFES.indBuffers.methodsxPort.

myLFES=calcTransporterIdxPort(myLFES): This function is analogous to calcMachineIdxPort() above, but instead calculates the index attributes in myLFES.transporters.methodsxPort.

myLFES=calcJH_KH_AH(myLFES): This function computes the transportation knowledge base (myLFES.JH), the transportation constraint matrix (myLFES.KH) and the transformation knowledge base (myLFES.AH).

myLFES=calcJHref_KHref_AHref(myLFES): This function computes the refined transportation knowledge base (myLFES.JHref), the refined transportation constraint matrix (myLFES.KHref) and the refined transformation system concept (myLFES.AHref).

myLFES=calcStrucDOF(myLFES): This function calculates the number of independent actions that completely define the number of available transformation, transportation and refined transportation processes in a system and assigns it to myLFES.DOFM, myLFES.DOFH, myLFES.DOFHref respectively.

myLFES=calcJS_KS_AS(myLFES): This function computes the system knowledge base (myLFES.JS), the system constraint matrix (myLFES.KS) and the system concept (myLFES.AS).
myLFES=calcAR(myLFES) : This function computes the Hetero-Functional Adjacency Matrix myLFES.AR and the projected Hetero-Functional Adjacency Matrix myLFES.ARproj. In parallel, the function calculates the number of physical continuity degrees of freedom and assigns their values to myLFES.DOFR1, myLFES.DOFR2, myLFES.DOFR3, and myLFES.DOFR4 for Types 1 through 4 respectively. Furthermore, it computes the number of functional sequence dependent degrees of freedom and assigns it to myLFES.DOFR5. Finally, the function computes the total degrees of freedom in the system and stores it as myLFES.DOFR.

myLFES = initializeControl(myLFES) : This function creates appropriately sized but empty sparse matrices to the controller agency matrix (myLFES.CAM), the controller adjacency matrix (myLFES.CADM) and the projected system adjacency matrix without services (myLFES.partialSAMproj).

myLFES=makeCAM(myLFES) : This function computes the controller agency matrix and assigns it to myLFES.CAM.

myLFES=makeCADM(myLFES) : This function computes the controller adjacency matrix and assigns it to myLFES.CADM.

myLFES=combineHFAMandCADM(myLFES) : This function computes the system adjacency matrix (without services) by combining the hetero-functional adjacency, the controller agency and the controller adjacency matrices. It is assigned to myLFES.partialSAMproj.

myLFES=makeServiceGraph(myLFES) : This function captures information regarding the different states involved in delivering an operand in the system and translates it into a service graph that defines the adjacency of the service activities in the system. For each service, it stores the positive incidence matrix, the negative incidence matrix and the dual-adjacency matrix under myLFES.services.MLpos, myLFES.services.MLneg and myLFES.services.dualAdjacency respectively.

myLFES=makeServiceFeasibility(myLFES) : This function computes the service feasibility matrix. It uses the structural degrees of freedom previously computed in the toolbox to compute the service degrees of freedom. It computes and assigns the following attributes to the myLFES object:

- myLFES.Lambda : Overall service feasibility matrix projected to DOFs.
- myLFES.xFormLambda : Service transformation feasibility matrix projected to transformation capabilities.
- myLFES.services.rawLambda : Service feasibility matrix in its original shape.
- myLFES.services.Lambda : Service feasibility matrix projected to DOFs.
- myLFES.services.xFormLambda : Service transformation feasibility matrix projected to system capabilities.
- myLFES.services.xPortLambda : Service transportation feasibility matrix.

myLFES=combineHFAMCADMService(myLFES) : This function combines the hetero-functional adjacency, controller agency, controller adjacency, and service graph matrices to produce the system adjacency matrix. This combined matrix is assigned to myLFES.SAMproj.

**Toolbox Validation**

The accurate production of a hetero-functional graph as an instantiated mathematical model is similar to that of the creation of a formal graph based exclusively upon nodes and edges. Formal graph toolboxes are able to create formal graphs as instantiated mathematical models because they reproduce the results that would be achieved for small systems computed by hand. In other words, these toolboxes serve to automate the construction of these mathematical models that would be impractical to do by hand. Consequently, the HFGT toolbox has been validated by making sure that its results match the results found in a number of early HFGT publications23, 24, 27, 32–34, 37, 44 where either manual or semi-automated methods were used. A wide variety of published test cases were tested so as to ensure that all of the HFGT toolbox data structures were instantiated and all of the HFGT toolbox methods were called.

**Conclusion**

This paper serves as a user guide to a recently developed Hetero-functional Graph Theory Toolbox. The toolbox is written in the MATLAB language and has been tested with v9.6 (R2019a). It is openly available on GitHub together with a sample input XML file for straightforward re-use. The paper details the syntax and semantics of the XML input, the myLFES (large flexible engineering system) data structure at the core of the toolbox and the functions used to construct and populate this
data structure. The toolbox has been fully validated against several peer-review HFGT publications. The development of a streamlined, computationally efficient, and openly-accessible toolbox that automates the underlying mathematical operations of HFGT enables the broader scientific community to apply HFGT to a wide variety of highly interconnected and heterogeneous engineering systems. Thus, this work enables several avenues for future research; particularly in the analysis, design, planning and operation of systems-of-systems.

References

1. De Weck, O. L., Roos, D. & Magee, C. L. *Engineering systems: meeting human needs in a complex technological world* (MIT Press, Cambridge, Mass., 2011).

2. Farid, A. M. An engineering systems introduction to axiomatic design. In Farid, A. M. & Suh, N. P. (eds.) *Axiomatic Design in Large Systems: Complex Products, Buildings & Manufacturing Systems*, chap. 1, 1–47 (Springer, Berlin, Heidelberg, 2016).

3. SE Handbook Working Group. *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities* (International Council on Systems Engineering (INCOSE), 2015).

4. Rumbaugh, J., Jacobson, I. & Booch, G. *The Unified Modeling Language Reference Manual* (Addison-Wesley, Reading, Mass., 2005).

5. Weilkiens, T. *Systems engineering with SysML/UML modeling, analysis, design* (Morgan Kaufmann, Burlington, Mass., 2007).

6. Friedenthal, S., Moore, A. & Steiner, R. *A Practical Guide to SysML: The Systems Modeling Language* (Morgan Kaufmann, Burlington, MA, 2011), 2nd edn.

7. Crawley, E., Cameron, B. & Selva, D. *System Architecture: Strategy and Product Development for Complex Systems* (Prentice Hall Press, Upper Saddle River, N.J., 2015).

8. Newman, M., Barabasi, A.-L. & Watts, D. J. *The structure and dynamics of networks* (Princeton University Press, 2006).

9. Barabási, A.-L. & Pósfaí, M. *Network science* (Cambridge University Press, Cambridge, 2016).

10. Xie, F. & Levinson, D. Modeling the growth of transportation networks: A comprehensive review. *Networks Spatial Econ.* 9, 291–307, DOI: 10.1007/s11067-007-9037-4 (2009).

11. Albert, R., Albert, I. & Nakarado, G. L. Structural vulnerability of the north american power grid. *Phys. review E* 69, 025103 (2004).

12. Amaral, L. A. N., Scala, A., Barthelemy, M. & Stanley, H. E. Classes of small-world networks. *Proc. national academy sciences* 97, 11149–11152 (2000).

13. Di Nardo, A., di natale, M., Santonastaso, G., Tzatchkov, V. & Yamanaka, V. Water network sectorization based on graph theory and energy performance indices. *ASCE J. Water Resour. Plan. Manag.* 140, –, DOI: 10.1061/(ASCE)WR.1943-5452.0000364 (2013).

14. Ivanov, D., Dolgui, A. & Sokolov, B. Applicability of optimal control theory to adaptive supply chain planning and scheduling. *Annu. Rev. control* 36, 73–84 (2012).

15. Yih, Y. *Handbook of healthcare delivery systems* (CRC Press, Boca Raton, FL, 2016).

16. Kivelä, M. et al. Multilayer networks. *J. complex networks* 2, 203–271 (2014).

17. Schoonenberg, W. C., Khayal, I. S. & Farid, A. M. *A Hetero-functional Graph Theory for Modeling Interdependent Smart City Infrastructure* (Springer, Berlin, Heidelberg, 2018).

18. Muhanji, S. O., Flint, A. E. & Farid, A. M. *eloT: The Development of the Energy Internet of Things in Energy Infrastructure* (Springer, Berlin, Heidelberg, 2019).

19. McFarlane, D., Sarma, S., Chirn, J. L., Ashton, K. & Wong, C. Y. Auto ID systems and intelligent manufacturing control. *Eng. Appl. Artif. Intell.* 16, 365–376 (2003).

20. McFarlane, D. et al. The Intelligent Product Driven Supply Chain. In *2002 IEEE International Conference on Systems, Man and Cybernetics*, vol. 4, 393–398 (2002).

21. McFarlane, D., Giannikas, V., Wong, A. C. & Harrison, M. Product intelligence in industrial control: Theory and practice. *Annu. Rev. Control.* 37, 69–88 (2013).

22. Meyer, G. G., Främling, K. & Holmström, J. Intelligent products: A survey. *Comput. Ind.* 60, 137–148 (2009).
23. Farid, A. M. & McFarlane, D. C. A Development of Degrees of Freedom for Manufacturing Systems. In IMS’2006: 5th International Symposium on Intelligent Manufacturing Systems: Agents and Virtual Worlds, 1–6 (Sakarya, Turkey, 2006).
24. Farid, A. M. Reconfigurability Measurement in Automated Manufacturing Systems. Ph.d. dissertation, University of Cambridge Engineering Department Institute for Manufacturing (2007).
25. Farid, A. M. Product Degrees of Freedom as Manufacturing System Reconfiguration Potential Measures. Int. Transactions on Syst. Sci. Appl. – invited paper 4, 227–242 (2008).
26. Farid, A. M. & McFarlane, D. C. Production degrees of freedom as manufacturing system reconfiguration potential measures. Proc. Inst. Mech. Eng. Part B (Journal Eng. Manuf. – invited paper 222, 1301–1314, DOI: 10.1243/09544054JEM1056 (2008).
27. Farid, A. M. Multi-Agent System Design Principles for Resilient Coordination and Control of Future Power Systems. Intell. Ind. Syst. 1, 255–269 (2015).
28. Thompson, D., Schoonenberg, W. C. & Farid, A. M. A Hetero-functional Graph Analysis of Electric Power System Structural Resilience. In IEEE Innovative Smart Grid Technologies Conference North America (Washington, DC, United states, 2020).
29. Lubega, W. N. & Farid, A. M. A Reference System Architecture for the Energy-Water Nexus. IEEE Syst. J. PP, 1–11 (2014).
30. Carey, C. & Farid, A. M. The energy-water nexus: A hetero-functional graph theory centrality analysis. Tech. Rep., Thayer School of Engineering at Dartmouth (2018).
31. Thompson, D. & Farid, A. M. Energy-water nexus: Reference architecture, hetero-functional graph theory analysis and equations of motion. Tech. Rep., Thayer School of Engineering at Dartmouth (2019).
32. Farid, A. M. Symmetria: Test Case for Transportation Electrification Research. Infrastructure Complex. 2, 1–10 (2015).
33. Farid, A. M. A Hybrid Dynamic System Model for Multi-Modal Transportation Electrification. IEEE Transactions on Control. Syst. Technol. PP, 1–12 (2016).
34. van der Wardt, T. J. & Farid, A. M. A hybrid dynamic system assessment methodology for multi-modal transportation-electrification. Energies 10, 653 (2017).
35. Schoonenberg, W. C. & Farid, A. M. A Dynamic Model for the Energy Management of Microgrid-Enabled Production Systems. J. Clean. Prod. 1, 1–10 (2017).
36. Schoonenberg, W. C. & Farid, A. M. A dynamic production model for industrial systems energy management. In 2015 IEEE International Conference on Systems Man and Cybernetics, 1–7 (Hong Kong, 2015).
37. Khayal, I. S. & Farid, A. M. Architecting a System Model for Personalized Healthcare Delivery and Managed Individual Health Outcomes. Complexity 1, 1–25 (2018).
38. Khayal, I. S. & Farid, A. M. The Need for Systems Tools in the Practice of Clinical Medicine. Syst. Eng. 20, 3–20 (2016).
39. Hegde, P., Schoonenberg, W. C., Thompson, D. & Farid, A. M. The hetero-functional graph theory toolbox. Tech. Rep., Laboratory for Intelligent Integrated Networks of Engineering Systems. Thayer School of Engineering at Dartmouth (2020).
40. MATLAB. version 9.6.0 (R2019a) (The MathWorks Inc., Natick, Massachusetts, 2019).
41. Harold, E. & Means, S. XML in a nutshell (O’Reilly Media, 2002).
42. Murata, T. Temporal uncertainty and fuzzy-timing high-level petri nets. In Billington, J. & Reisig, W. (eds.) Application and Theory of Petri Nets 1996, vol. 1091 of Lecture Notes in Computer Science, 11–28, DOI: 10.1007/3-540-61363-3_2 (Springer Berlin Heidelberg, 1996).
43. Falkena, W. xml2struct (MATLAB Central File Exchange., 2020).
44. Abdulla, H. & Farid, A. M. Extending the energy-water nexus reference architecture to the sustainable development of agriculture, industry & commerce. In First IEEE International Smart Cities Conference, 1–7 (Guadalajara, Mexico, 2015).

Author contributions statement
Prabhat Hegde is the lead author on this paper and contributed to the development of the hetero-functional graph theory toolbox. Wester C.H. Schoonenberg is the lead developer of the hetero-functional graph theory toolbox and contributed to the writing of this paper. Dakota Thompson is the secondary developer of the hetero-functional graph theory and contributed to the writing.
of this paper. Amro M. Farid architected the hetero-functional graph theory toolbox and managed the entirety of the project including the writing of this paper.