Mechanical Systems and Signal Processing 148 (2021) 107144

Contents lists available at ScienceDirect

Mechanical Systems and Signal Processing

journal homepage: www.elsevier.com/locate/ymssp

Foundations of Population-based SHM, Part II: Heterogeneous populations – Graphs, networks, and communities

J. Gosliga, P.A. Gardner, L.A. Bull, N. Dervilis, K. Worden *

Dynamics Research Group, Department of Mechanical Engineering, University of Sheffield, Mappin Street, Sheffield S1 3JD, UK

ARTICLE INFO

Article history:
Received 23 December 2019
Received in revised form 17 April 2020
Accepted 15 July 2020
Available online 4 August 2020

Keywords:
Population-based structural health monitoring (SHM)
Irreducible Element (IE) model
Attributed Graph (AG)
Complex networks of structures

ABSTRACT

This paper is the second in a series of three which aims to provide a basis for *Population-Based Structural Health Monitoring* (PBSHM); a new technology which will allow transfer of diagnostic information across a population of structures, augmenting SHM capability beyond that applicable to individual structures. The new PBSHM can potentially allow knowledge about normal operating conditions, damage states, and even physics-based models to be transferred between structures. The first part in this series considered *homogeneous* populations of nominally-identical structures. The theory is extended in this paper to *heterogeneous* populations of disparate structures. In order to achieve this aim, the paper introduces an abstract representation of structures based on *Irreducible Element* (IE) models, which capture essential structural characteristics, which are then converted into *Attributed Graphs* (AGs). The AGs form a *complex network* of structure models, on which a metric can be used to assess structural similarity; the similarity being a key measure of whether diagnostic information can be successfully transferred. Once a pairwise similarity metric has been established on the network of structures, similar structures are clustered to form *communities*. Within these communities, it is assumed that a certain level of knowledge transfer is possible. The transfer itself will be accomplished using machine learning methods which will be discussed in the third part of this series. The ideas introduced in this paper can be used to define precise terminology for PBSHM in both the homogeneous and heterogeneous population cases.

1. Introduction

This paper is the second in a sequence of three which aims to lay the foundations of *Population-Based Structural Health Monitoring* (PBSHM). In the first part [1], a theory for homogeneous populations of nominally-identical structures was presented, and a new machine-learning methodology, based on the idea of a population *form*, was demonstrated. In this paper, the idea of PBSHM will be extended to *heterogeneous* populations containing disparate structures. However, some discussion will first motivate the development of the new technology.

By going beyond routine visual inspection, *structural health monitoring* (SHM) has the potential to reduce both the cost of maintenance and the frequency of structural failures; it achieves this by continuous monitoring of condition and performance of structures using permanently-installed sensors. Data-based SHM [2] achieves its objectives via machine learning operations on measured data, and can move through various levels of diagnostic capability – detection, location, quantifica-

* Corresponding author.
E-mail address: k.worden@sheffield.ac.uk (K. Worden).

https://doi.org/10.1016/j.ymssp.2020.107144
0888-3270/

tion [3] – if data representing the various conditions are available. If physics-based models of damage progression are available, SHM can potentially move to prognostics – determination of safe residual life [2].

Unfortunately, the reality is that data – damage-state data in particular – are often scarce or unavailable. Moving to PBSHM offers a means of alleviating this problem by providing methods for transferring diagnostic capability between members of populations of structures; in particular, if damage-state data are available for one structure, they can be used to improve diagnostic power for other structures. Furthermore, if the structures concerned are similar enough, the technology might be used to increase the relevance of physics-based models for entire populations, thus leading to improved prognostics.

One of the simplest contexts for PBSHM concerns populations of structures where all the structures are *nominally identical*, and subjected to the same set of boundary and environmental conditions – with the term *nominally identical* implying that the only variations between the structures arise due to manufacturing differences. Such a population is described in [1] as a *strongly-homogeneous* population. The paper shows that knowledge transfer is possible across the population via the construction of a model – the *form* – which captures both the generic structure behaviour, and the characteristic variations across the population.

Earlier work on homogeneous populations [4,5], exploited the fact that nominally-identical structures should exhibit the same behaviour, given similar boundary conditions, by attempting to use the model for an individual wind turbine to predict the behaviour of another in the same wind farm. This strategy provides a normal condition model which is robust to certain environmental conditions, as the boundary conditions imposed by the wind speed and temperature should be relatively consistent across the wind farm. Therefore, if a particular wind turbine starts to exhibit behaviour which is inconsistent with the other turbines in the farm, this could represent damage. Further work in [1], introduced the first version of the *form*, to represent all of the individuals in a homogeneous population.

The question then arises: if knowledge transfer is possible between nominally-identical structures, what about structures which are not the same, yet share *some* degree of similarity? For example, a passenger jet and a single seat propeller aircraft are clearly very different aeroplanes; however, they both share common features: wings, a fuselage, landing gear, etc. Aeroplanes are also often subjected to similar operating conditions, and hence boundary conditions. Intuition would suggest that there must be some level of knowledge transfer possible between these two structures. However, merely having a sense that knowledge transfer is possible is not sufficient, a method for quantifying the similarities and differences between structures is required. This paper will develop a representation of structures which makes it possible to assess the similarity of structures.

The abstract representation of structures developed in this paper is designed firstly to quantify the degree of similarity between structures, and secondly to facilitate the transfer of inferences via machine learning. The framework is primarily for populations which do not contain nominally-identical structures; these are termed *heterogeneous* populations. This is an informal definition, which will be made more precise at the end of the paper. One of the aims of the framework here is to identify if non-identical structures contain 'common substructures' so that knowledge transfer is still valid at the substructure level. For example, in most cases, a bridge and an aeroplane do not share any common features; however, the propellor of an aeroplane and the blades of a wind turbine may share similar behaviour, which allows transfer of damage detection and location capability.

The abstract representation introduced in this paper requires two stages. In the first stage, an *irreducible element* (IE) model is constructed, which decomposes the structure into components which are deemed to have well-established dynamic behaviour e.g. beams, plates and shells; the model is not designed for dynamic prediction, but rather to identify the essential elements which fix the nature of the structure. At the second stage of the process, the IE model is converted into an *attributed graph* (AG), which is the final abstraction of the structure. AGs have been used successfully to quantify structural similarity in other fields – most notably chemistry, bioinformatics and computer science [6]. In manufacturing, AGs have been used to measure the degree of similarity in the geometry of manufactured parts [7,8]. The *attributes* of the nodes and edges of the graphs are parameter sets associated with the graph elements which describe aspects – such as the topology, materials and geometry – which are relevant in fixing the dynamic behaviour of the respective structure. To provide a measure of similarity between the topology of two AGs – and hence between the structures that they represent, algorithms are employed here which find the largest subset of nodes and edges common to the two graphs (*maximum common subgraph*) [9,10]. Physically, this procedure represents finding the largest substructure that the two structures have in common. The attributes of the AGs will also partly determine the overall similarity metric. The maximum common subgraph between two AGs will represent the largest common substructure between the two structures in question.

This paper will also describe the *network* of structures, formed from the AGs of various structures. The network is a space where the distance between members of a heterogeneous population is determined by the degree of similarity; similar structures will be 'closer together'. Clustering algorithms can then be used to create *communities* of similar structures. Within these communities, there is a greater probability that knowledge transfer between structures is possible; e.g. within a community of aeroplanes, some information regarding damage states for wings should be transferable, as all aeroplanes have wings.

The actual mechanics of knowledge transfer are beyond the scope of this paper, and will be discussed in the final part of this series [11]. The methodology will build on previous work showing that *transfer learning* can be applied in an SHM context [12]. The main aim of the current paper is to construct the framework for assessing if transfer is likely to be valuable.

The layout of the paper is as follows: Section 2 will introduce the irreducible element (IE) model, and discuss which ingredients might be necessary in capturing the essential dynamic nature of a structure. Section 3 will show how the IE model is converted into an attributed graph (AG); this section depends on a number of results from graph theory, which are summarised for the convenience of the reader. Section 4 shows how the AG can be used to establish a degree of similarity between structures, by identifying the *maximum common subgraph* of the AG representations; again, the necessary graph theory is presented, including the fundamental algorithm applied here – the *Bron-Kerbosch* algorithm. Section 6 extends the discussion of similarity measures to include attributes of the graphs, which are essential in refining the measures beyond simple topology. At this point, enough theory has been developed, that formal definitions of the main concepts in PBSHM can be made precise. Finally, conclusions are presented.

2. Irreducible element representations of structures

To determine whether or not two structures are similar enough to allow knowledge transfer, it is unnecessary to consider every property or dimension of the structure. For example, to determine geometric similarity (to a certain resolution), it would be computationally inefficient to compare 3D models (such as Finite Element (FE) or CAD models) of structures directly. Such 3D models contain a large amount of redundant information (i.e. the detailed construction of the mesh in an FE model) that have no strong effect on the overall geometry of the structure. It is more efficient to consider only the properties and dimensions of a structure which significantly effect knowledge transfer.

The proposed solution is to abstract the significant properties and dimensions to create an *Irreducible Element* (IE) representation of the structure. For a given structure, an IE model attempts to simplify the geometry of various structural components into shapes that have well-defined dynamic behaviour, such as beams or plates, as illustrated in Fig. 1. The IE representation can be used to determine which inference tools are appropriate, as well as the level of information that can be inferred.

The motivation for using shapes that have well-defined dynamic behaviour is the belief that since, for example, all plates exhibit similar behaviour (given a similar set of boundary conditions), they would respond to damage in a similar way, or would at least have similar damage-sensitive features. In the case where two IEs have the same shape and dimensions, it can be assumed that the behaviour exhibited is identical for a given set of boundary conditions.

As well as simplifying the geometry, the IE representation also captures important information about the topology, and materials in a structure. This information makes it possible to determine the level of homogeneity between two structures.

The geometry, topology, and materials of a structure tie into the level of knowledge transfer between two structures. The labels used in knowledge transfer are based on Rytter's hierarchy [3]. (The relation between level of inference possible and the differences in the geometry, topology, and materials – captured in the IE representation of a structure – are discussed further in [11].)

Geometry is important when determining the overall dynamic behaviour of structures. If it is possible to assume that the dynamic behaviour is the same between two structures, then (given the same environmental conditions) any difference in the behaviour of the two structures must be due to damage. This can be seen in the Lillgrund windfarm example [5].

If the topology of two structures is the same, then it is possible to assume that any damage location labels can be applied in a consistent manner on both of them. This observation is true even if the entire structures are not similar. If two structures share a common substructure, then it may be possible to infer damage location between the two structures, provided damage occurs within this common substructure. This idea works, because within this common substructure, feature and location labels should be consistent.

Including materials in the description means one can determine whether inferences on the type of damage is possible. If two structures are made from the same materials, then the extent and classification labels for damage will be consistent, enabling the use of transfer learning. The following sections will detail the relevance of these properties further.

2.1. Geometry

Since it may not be possible to obtain complete geometrical information for all structures, it is necessary to define a hierarchy of properties which define the IE to increasing levels of precision. A reduced hierarchy of properties can be seen in Tables 1 and 2, where the coarsest level of description is the *geometry* class, followed by *shape*. In the full IE model, there will be a further level in the hierarchy of properties, where the shape is fully defined with the major dimensions – for example length, breadth and width. This hierarchy of properties encodes the fact that certain transfer learning approaches are valid at a coarser level than others, with some able to deal with different parameters in the source and target domains when transfer learning is applied [11]. Similarities become better defined at a lower level of the hierarchy and improve the chance of success of knowledge transfer: using the example of a beam, certain inferences will be possible across all beams, some will require that the cross-section is the same, and other inferences rely on the geometry matching exactly.

The first four proposed geometry classes are: *beam*, *plate*, *shell*, and *complex* (and these will be sufficient for the purposes of the discussion within this paper). The beam and shell are fairly self-explanatory and correspond to the definitions used in multi-body modelling. The plate class is introduced to differentiate between shells which enclose a volume and those that do not. The complex geometry class allows flexibility in how the IE is defined.

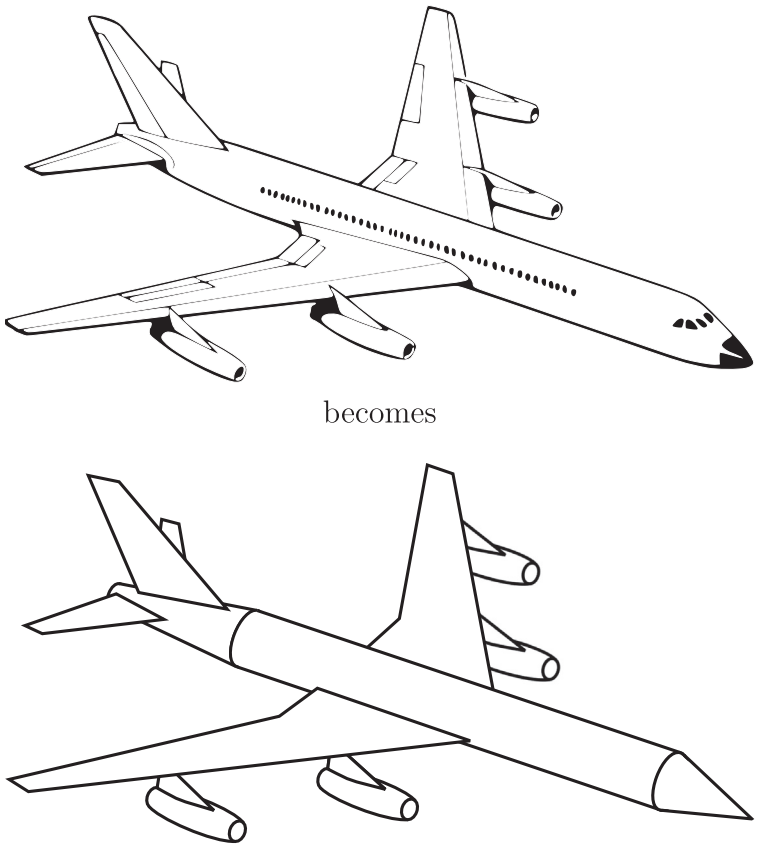

Fig. 1. Irreducible Element representations seek to remove dimensions from a model which do not significantly affect the bulk dynamic properties of a structure, or have no relevance in determining the level of knowledge transfer possible.

Table 1
List of elements and their properties for Turbine 1. FRP here stands for Fibre-Reinforced Polymer.

Wind Turbine					
Name	Element ID	Material Class	Material	Geometry	Shape
Rotor blade	A	Composite	FRP	Beam	Aerofoil
Rotor blade	B	Composite	FRP	Beam	Aerofoil
Rotor blade	C	Composite	FRP	Beam	Aerofoil
Rotor hub	D	Composite	FRP	Complex	Rotor hub
Nacelle	E	Composite	FRP	Shell	Cuboid
Tower Section 1	F	Metal	Steel	Beam	Cylindrical
Tower Section 2	G	Metal	Steel	Beam	Cylindrical
Tower Section 3	H	Metal	Steel	Beam	Cylindrical
Foundation	I	Ceramic	Concrete	Plate	Cylindrical
Name	Element ID	Boundary	–	–	–
Footing	1	Ground	–	–	–

There are three main cases where the complex class becomes necessary. Firstly, the *complex* class is useful when there exists an element which is not easily described by a simple shape, for example the rotor hub on a wind turbine in Table 1. Secondly, the complex class can be used in the case where it is beneficial to represent a collection of components as a single element, for example the landing gear in Table 2 is represented as a single complex element. Thirdly – and finally – the complex class is used to represent the components within the structure that could in theory be broken down further into simpler IEs[1], but doing so would mean creating multiple elements out of a single structural component.

[1] There might appear to be a contradiction in the terminology here, in the idea that an *irreducible* element might be decomposed. However, the term 'irreducible' is used here in the sense that an IE model can provide a description of *sufficient* complexity for the purposes of matching structures, but the user may choose a higher resolution.

Table 2
List of elements and their properties for Aeroplane 1.

Element designations for Aeroplane 1					
Name	Element ID	Material Class	Material	Geometry	Shape
Fuselage	A1	Composite	FRP	Shell	Truncated cone
Fuselage	A2	Composite	FRP	Beam	Cylindrical
Fuselage	A3	Composite	FRP	Shell	Cone
Wing 1	B	Composite	FRP	Beam	Aerofoil
Pylon 1	C	Composite	FRP	Complex	Pylon
Engine 1	D	Assembly	Assembly	Shell	Cylinder
Pylon 2	E	Composite	FRP	Complex	Pylon
Engine 2	F	Assembly	Assembly	Shell	Cylinder
Wing 2	G	Composite	FRP	Beam	Aerofoil
Pylon 3	H	Composite	FRP	Complex	Pylon
Engine 3	I	Composite	Assembly	Shell	Cylinder
Pylon 4	J	Composite	FRP	Complex	Pylon
Engine 4	K	Assembly	Assembly	Shell	Cylinder
Vert stabiliser 1	L	Composite	FRP	Beam	Aerofoil
Vert stabiliser 2	M	Composite	FRP	Beam	Aerofoil
Horz stabiliser	N	Composite	FRP	Beam	Aerofoil
Front landing gear	O	Assembly	Assembly	Complex	Assembly
Rear landing gear	P	Assembly	Assembly	Complex	Assembly
Name	Element ID	Boundary	–	–	–
Tarmac	1	Ground	–	–	–

An example of where a complex geometry has been split into simpler elements can be seen in Table 2, where the fuselage of the aeroplane has been broken into separate elements in order to better define the geometry. However, when it comes to similarity matching, there are some cases where it would be better to define the fuselage as a single complex element so that each aeroplane now shares a common feature: a fuselage element attached to two wing elements.

The complexity of the geometry and the degree of uncertainty within the match will influence the certainty with which inferences can be made. Within transfer learning, a classifier or regression model trained for a task in one domain (source) is used in another domain (target). An example of transfer learning is when pre-trained convolutional neural networks are applied to a wide range of image recognition problems. However, transfer learning only works if the source and target domains are sufficiently similar. Therefore, if transfer learning approaches are used to make inferences within SHM, the structures would need to be similar, otherwise *negative transfer* may arise. In negative transfer, differences in the source and task domain may cause mislabelling, for example, labelling normal condition data as damage data and vice versa. The concepts of transfer learning and negative transfer are discussed in [11]. Where the geometry is complex, avoiding negative transfer may not be guaranteed and caution must be exercised.

2.2. Topology

The topology of the structure plays a large part in determining the dynamic behaviour of a structure. The topology here is determined by physical connections between the various elements, as shown in Fig. 2. With reference to Rytter's hierarchy [3], the topology has a strong link to damage location. Without corresponding topology, structures cannot have fully consistent location labels.

The *complex* class provides flexibility to simplify the topology of structures where it is believed that the topology is not significant for the problem. For example, if the application was comparing the overall dynamics of two different aeroplanes, it may be useful to combine the landing gear or the fuselage into a single element so that differences in overall topology (configuration), such as engine placement, are emphasised over local topology; where local topology refers to locations within either the fuselage or landing gear. Alternatively, if the local topology was relevant to the application, such as locating damage within the fuselage, it would be beneficial to break this complex element into constituent parts.

2.2.1. Joints

Joints are classed as capturing topological information since they define where connections between elements occur; they also have the same effect on the label consistency as topology [11]. Example lists of joints and their properties can be found in Tables 3 and 4 for a wind turbine and an aeroplane respectively. The joints contain properties that are important both for constructing the attributed graph, as well as for creating physical models of the structure.

The element set describes the set of elements which are connected by the joint; it is therefore crucial in recovering the topology of the structure.

The coordinates define the geometrical location of the joint; the joint coordinate aids in constructing physical models from the IE representation. These coordinates define the mid-point of the joint and are determined by examining the geometry of the structure.

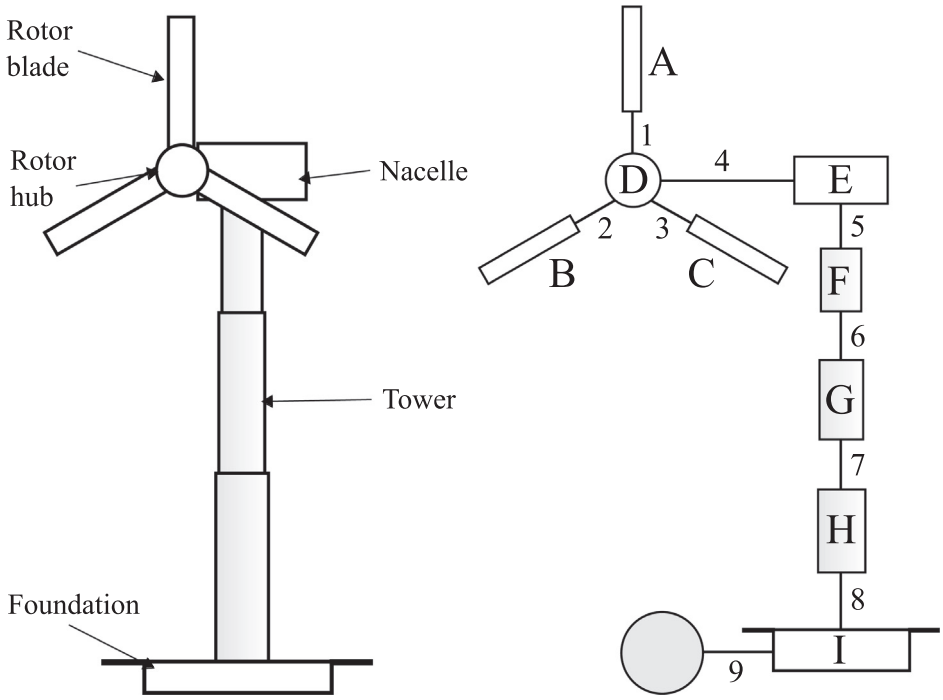

Fig. 2. An expanded IE representation of a wind turbine with the elements labelled A to I and the connections labelled 1 to 9. The shaded node is a special element representing the ground, where the boundary condition is defined in the attributes for Joint 9.

Table 3
List of joints and their properties for Turbine 1.

Wind Turbine						
Joint ID	Element set	Coordinate	Class	Type	Disp. DoF	Rot. DoF
1	{A, D}	(8, 15, 235.75)	Kinematic	Bearing	[x, y, z]	[y, z]
2	{B, D}	(8, 14, 254)	Kinematic	Bearing	[x, y, z]	[y, z]
3	{D, E}	(10, 15, 253)	Kinematic	Bearing	[x, y, z]	[y, z]
4	{D, C}	(8, 16, 254)	Kinematic	Bearing	[x, y, z]	[x, y]
5	{E, F}	(15, 15, 250)	Kinematic	Bearing	[x, y, z]	[x, y]
6	{F, G}	(15, 15, 183))	Static	Bolted	–	–
7	{G, H}	(15, 15, 105)	Static	Bolted	–	–
8	{H, I}	(15, 15, 5)	Static	Bolted	–	–
9	{I, 1}	(15, 15, 0)	Static	Clamped	–	–

The joint type describes some of the key physical aspects of the joint, such as the type of joint and the degrees of freedom that are constrained. The joint type and restricted degrees of freedom are again hierarchical, as it may not be possible to obtain this information in every case. It should also be noted that for static joints, it is assumed that all degrees of freedom are constrained and so, for static joints, this property is not specified.

Anticipating the next section a little, the *nodes* in the attributed graphs will represent the IEs, the *edges* will represent the joints between them.

2.3. Materials

The topology and geometry of the structure are most strongly linked with the transfer of damage detection and damage location labels. The material properties, on the other hand, determine the assessment and classification labels.

A hierarchical set of material properties are used to describe the materials within an IE, as again it cannot be guaranteed that detailed information on materials used will be available for each structure. Within the material properties, the coarsest level of detail will be the *material* class (ceramics, metals). The next level of detail will be a specific description of the material within the class, for example brass and steel belong to the material class of metals. The finest level of detail will be the specific properties of the material, for example Young's Modulus and density. Also included in material properties may be additional information such as material grade. Once again, to be confident of an exact match between two structures, all

Table 4
List of joints and their properties for Aeroplane 1.

Joint designations for Aeroplane 1						
Joint ID	Element set	Coordinate	Class	Type	Disp. DoF	Rot. DoF
1	{A1, A2}	(34.2, 14.68, 5.165)	Static	Perfect	–	–
2	{A2, A3}	(34.2, 60.96, 5.165)	Static	Perfect	–	–
3	{A2, B}	(32.2, 29.79, 2.89)	Static	Lug	–	–
4	{B, C}	(13.2, 42.67, 4.74)	Static	Complex	–	–
5	{C, D}	(13.2, 40.17, 4.74)	Static	Complex	–	–
6	{B, E}	(23.2, 30.79, 3.57)	Static	Complex	–	–
7	{E, F}	(23.2, 28.29, 3.57)	Static	Complex	–	–
8	{A2, G}	(36.2, 29.79, 2.89)	Static	Lug	–	–
9	{G, H}	(45.2, 30.79, 3.57)	Static	Complex	–	–
10	{H, I}	(45.2, 28.29, 3.57)	Static	Complex	–	–
11	{G, J}	(55.2, 42.67, 4.74)	Static	Complex	–	–
12	{J, K}	(55.2, 40.17, 4.74)	Static	Complex	–	–
13	{A3, L}	(33.2, 68.58, 7.55)	Static	Lug	–	–
14	{A3, M}	(35.2, 68.58, 7.55)	Static	Lug	–	–
15	{A3, N}	(34.2, 64.58, 9.16)	Static	Lug	–	–
16	{A1, O}	(34.2, 7.75, 1.75)	Static	Complex	–	–
17	{A2, P}	(34.2, 29.67, 1.75)	Static	Complex	–	–
18	{O, 1}	(34.2, 7.75, 0)	Kinematic	Plane	[z]	[x, y]
19	{P, 1}	(34.2, 29.67, 0)	Kinematic	Plane	[z]	[x, y]

of the material properties must be known down to the bottom of the hierarchy, otherwise there will be some degree of uncertainty.

The material properties determine whether it is possible to make inferences using damage assessment and classification labels between two structures. Two materials of the same material class will experience similar failure modes, and could be expected to exhibit a similar response to a particular type of damage giving more confidence in the classification of damage. However, the more similar the materials, the more likely it is that the assessment (extent) of the damage will be the same between two structures. For example, materials of the same class, such as aluminium and steel, will both suffer corrosion (damage classification). However, there is little guarantee that the change in material properties will be the same for the same extent of corrosion (damage assessment). If the materials are identical, then transferring damage assessment and classification labels is trivial.

2.4. Boundary conditions

Boundary conditions are essentially a type of joint, and are defined using special edges, which determine the constrained degrees of freedom. In fact, a complete boundary condition is specified by an edge-node pair; defining the boundary condition and the physical cause of the boundary condition.

Boundary conditions imposed by the same root cause are represented by the same node. The justification for this is partially physical – all connections to 'ground' are modelled as rigid connections to the earth – but also practical. Having single edge-node combinations for boundary conditions makes it possible to calculate the number of connections to any particular boundary condition by simply examining the number of edges connected to that specific boundary condition node.

In the case of the aeroplane described in Tables 2 and 4, both sets of landing gear are restricted by the tarmac (ground), as the aeroplane sits on the runway.[2] As such, the nodes for both sets of landing gear connect to the same boundary condition node. (In this particular case, both boundary conditions can also be represented using the same joint.)

The ground also features as part of a boundary condition for the wind turbine in Table 1, although as can be seen from Table 3, the wind turbine has a permanent static connection to ground. The same type of boundary condition (ground) can manifest in different ways depending on how the structure is connected. Therefore, a complete match in boundary conditions requires that both the type of node, and the joint attributes carried by the edge connected to the boundary, are the same.

3. Producing an attributed graph

For the attributed graph (AG) the same information is embedded as for the IE representation, but whereas the properties for the IE representation are organised in a tabular format to improve readability by humans, the information in the AG is

[2] If the aircraft is stationary, this joint represents static friction; if the aircraft is in motion, one would need to consider the joint as a rolling friction; if the aircraft were in flight, both the node and joint would disappear.

organised so that it can be more efficiently processed by a graph-matching algorithm. The topology of the graph itself is extracted from the properties in the element and joint tables which define the IE representation.

3.1. Definitions

A graph G is defined by a set of nodes V and edges E, and so $G = (V, E)$ [13]. The edge and node set of a graph can alternatively be represented as $E(G)$ and $V(G)$, respectively; this is useful in contexts where it is necessary to define the vertex and edge sets for multiple graphs.

An *attributed graph* is obtained when the graph structure encodes additional information beyond its topology. In this case, nodes and/or edges can each have sets of parameters associated with them.

It is also possible to define a graph by its adjacency list. The adjacency list contains each node in V along with its neighbourhood, where the neighbourhood N for each node is defined as $N(v) = \{u \in V | (u, v) \in E\}$. An example of a graph and the corresponding adjacency list is shown in Fig. 3. The neighbourhood N as defined, does not contain the node itself; this is known as the *open neighbourhood* of v. The *closed neighbourhood* of a node v, contains the node itself and is defined as $N[u] = N(v) \cup v$. Neighbourhoods are assumed to be open unless otherwise specified.

If two nodes from $V(G)$, say v_1 and v_2, are adjacent, then $(v_1, v_2) \in E(G)$.

If there is a path from any node in G to any other node in G, then the graph is *connected*.

For the graph defined in Fig. 3: curly brackets are used to specify that the data type used is 'dictionary'; the quotation marks are used to denote strings, which are used as node labels in the code; and the square brackets are used to specify a 'list'. The edges of the graph can be obtained by examining the dictionary 'key' (to the left of the colon) and creating a pair with any single entry in the list of neighbours (to the right of the colon). For example, the edge (A, D) can be obtained by pairing the dictionary key A with its neighbour D. Alternatively, the equivalent edge (D, A) could be obtained by pairing the dictionary key D with its neighbour A. In general, the attributes associated with nodes and edges will also require specification of parameter vectors $\underline{\theta_i}$ and $\underline{\theta_{ij}}$, which capture numerical values e.g. values associated with material constants, or Euler angles showing the orientation of joints.

The dictionary data type (again Python is used for illustration here) is also used to store the properties from the IE representation as node and edge attributes. This data type allows the graph-matching algorithm to easily query the attributes of a given node, since the node label can be used as a dictionary key. When creating a dictionary to store node attributes, the element ID is used as the dictionary key, and the node attributes are stored as a list, with nested lists containing the geometrical and material attributes.

The node attributes for the elements in Table 1 appear as:

turbine.elements =	{'A'	:	[['Composite', 'FRP'],	['Beam', 'Aerofoil']],
	'B'	:	[['Composite', 'FRP'],	['Beam', 'Aerofoil']],
	'C'	:	[['Composite', 'FRP'],	['Beam', 'Aerofoil']],
	'D'	:	[['Composite', 'FRP'],	['Complex', 'Rotor hub']],
	'E'	:	[['Composite', 'FRP'],	['Shell', 'Cuboid']],
	'F'	:	[['Metal', 'Steel'],	['Beam', 'Cylindrical']],
	'G'	:	[['Metal', 'Steel'],	['Beam', 'Cylindrical']],
	'H'	:	[['Metal', 'Steel'],	['Beam', 'Cylindrical']],
	'I'	:	[['Composite', 'Concrete'],	['Plate', 'Cylindrical']],
	'1'	:	[['Ground'] }	

For creating a dictionary of edge attributes, the joint ID is exchanged with the element set. The reason for exchanging the joint ID and element set, is that when looking up edges in resulting subgraphs, the easiest way to query if an edge exists between node $v_1$and node $v_2$is to check whether $(v_1, v_2) \in E$; this naturally leads to edges being labelled by their node pair. The rest of the edge attributes (for the turbine joints in Table 3) is then organised in a similar fashion to the node attributes, where hierarchical attributes are contained within nested lists:

turbine.joints =	{('A', 'D')	:	['1',	[8, 15, 235.75],	[['Kinematic', 'Bearing'],	[['x', 'y', 'z'],	['y', 'z']]],
	('B', 'D')	:	['2',	[8, 14, 254],	[['Kinematic', 'Bearing'],	[['x', 'y', 'z'],	['y', 'z']]],
	('C', 'D')	:	['3',	[8, 16, 254],	[['Kinematic', 'Bearing'],	[['x', 'y', 'z'],	['y', 'z']]],
	('D', 'E')	:	['4',	[10, 15, 253],	[['Kinematic', 'Bearing'],	[['x', 'y', 'z'],	['x', 'y']]],
	('E', 'F')	:	['5',	[15, 15, 250],	[['Kinematic', 'Bearing'],	[['x', 'y', 'z'],	['x', 'y']]],
	('F', 'G')	:	['6',	[15, 15, 183],	[['Static', 'Bolted']],		
	('G', 'H')	:	['7',	[15, 15, 105],	[['Static', 'Bolted']],		
	('H', 'I')	:	['8',	[15, 15, 5],	[['Static', 'Bolted']],		
	('I', '1')	:	['9',	[15, 15, 0],	[['Static', 'Fixed']] }		

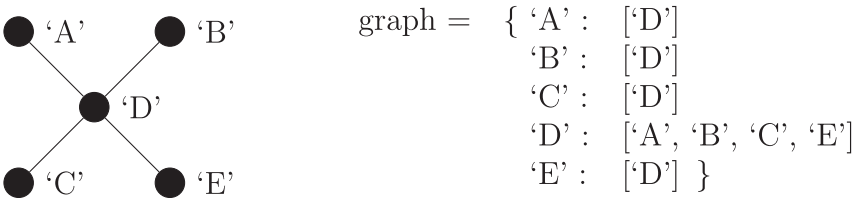

Fig. 3. A graph and its corresponding adjacency list. In this particular case, the adjacency list has been represented using a Python dictionary.

where the regular parentheses represent a *tuple* (essentially an ordered list – a common structure in modern computer languages). While organising the edge attributes in this way reduces the readability from a human perspective, it makes it easier for a graph-matching algorithm to search for node or edge attributes. Furthermore, there are no empty rows or columns when presented with an incomplete set of attributes.

Given the set of nodes *V*and edges *E* for a structure, it is possible to plot the graph, as shown in Fig. 4. Plotting the graphs is not necessary for graph comparison, but can aid in assessing whether or not the code is producing valid subgraphs.

Once an AG is defined, it immediately induces a basic modal model – the *Canonical Modal Model* (CMM). This is constructed as a lumped-mass model as follows: first, one replaces each node (IE) in the model by a lumped mass computed using the appropriate node attributes (i.e. density and dimensions). Secondly, one replaces each edge by a spring with stiffness computed using the edge attributes. Taking care to preserve ground nodes, this procedure defines a basic model from which modal invariants (natural frequencies) can be computed.

4. Determining similarity using attributed graphs

An IE representation of a structure is a way of abstracting and representing aspects of a structure that are important for successful knowledge transfer. By comparing the IE representations of two structures, it is possible to determine the level of inference possible between them. To allow this comparison to be performed automatically, it is more efficient to restructure the information into the form of an AG.

At the risk of repetition, the IE model and AG encode the same structural information; the IE model is more easily visualised, while the AG is more easily parsed by the PBSHM algorithms. Generating the AG from the IE representation of a structure is described in the previous section; this process is also illustrated in Fig. 5.

In what follows, it is assumed that the steps described in the previous sections have been followed, generating the AGs for an aeroplane and a wind turbine as shown in Fig. 4. The attributes for these AGs are taken from the IE representations, where the elements and joints for the aeroplane come from Tables 2 and 4 respectively, and the elements and joints for the wind turbine come from Tables 1 and 3.

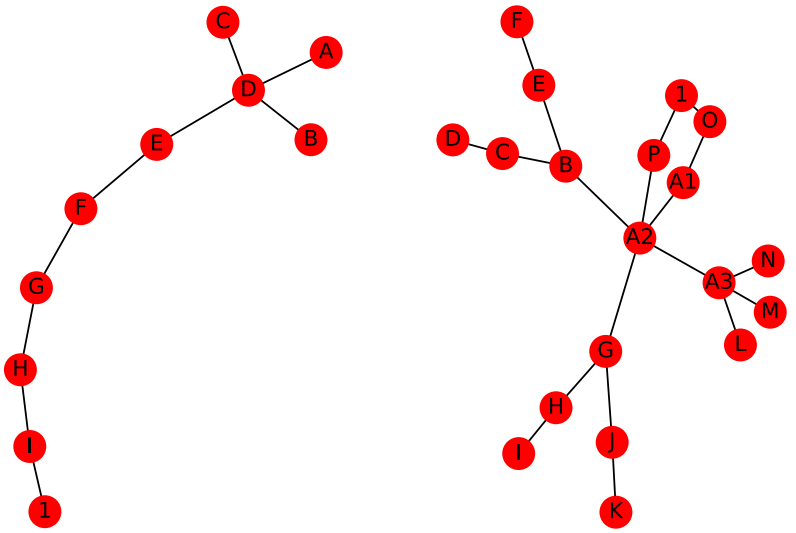

Fig. 4. The graphs produced from list of elements in Tables 2 and 1 and list of joints in Tables 4 and 3. The graph shown on the left is from the wind turbine (compare with Fig. 2) and the graph on the right is fro.m the aeroplane.

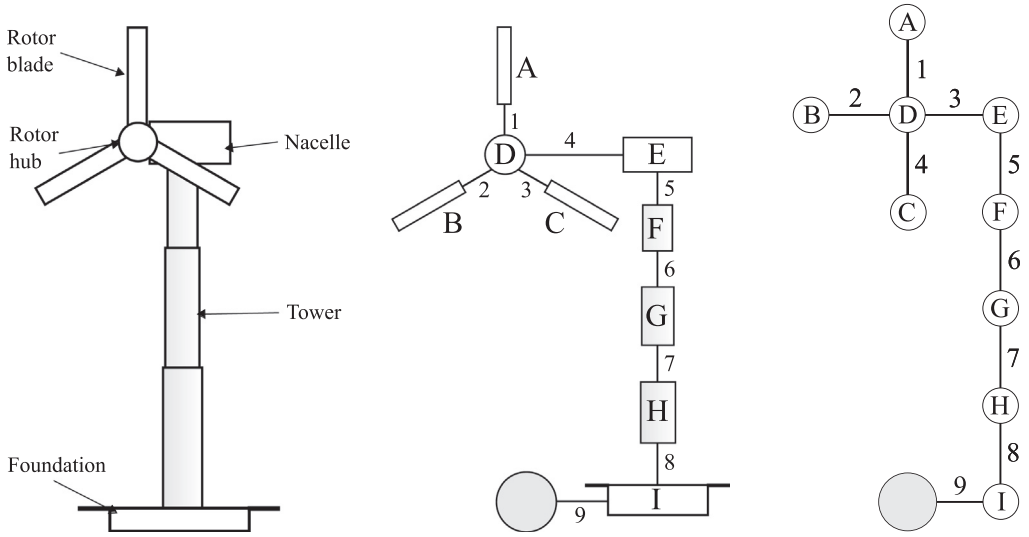

Fig. 5. The evolution of an attributed graph. The structure in this case is a wind turbine, and shown (from left to right) are the IE model; an expanded version of the IE model, showing the element and joint designations; and finally, the AG.

4.1. Introduction to graph-matching algorithms

The level of similarity between two structures is determined by the size of the largest substructure between them that can be considered to have some level of homogeneity. For example, when comparing two structures within a homogeneous population, the largest substructure that they have in common will be the entire structure. For two structures in a heterogeneous population, these substructures will be smaller; for example, the propellers on certain aeroplanes may be considered as having some level of homogeneity.

Since structures are represented as graphs, their substructures are represented by subgraphs. Therefore, the problem of finding the largest common substructure between two structures is reduced to finding the largest common subgraph between their respective attributed graphs.

There exist several algorithms for finding the largest common subgraph between two parent graphs, but in order to describe these algorithms effectively, it is necessary to provide several further definitions.

4.1.1. Isomorphism

Two graphs, G and H, are said to be isomorphic if there exists a one-to-one mapping from $V(G) \rightarrow V(H)$, and a one-to-one mapping from $E(G) \rightarrow E(H)$. Isomorphic graphs have the same topology, but have different node labels. Therefore, if two structures have the same topology, then they can be represented by isomorphic graphs. Since all structures within a homogenous population have the same topology, their AGs will be isomorphic.

4.1.2. Subgraphs

A graph G' is said to be a subgraph of G, if $G' \subseteq G$, which implies $V' \subseteq V$ and $E' \subseteq E$.

There are different types of subgraph. The types of subgraphs are differentiated by the rules for generating the subset of edges E', given the subset of nodes V'. For example, given an arbitrary subset of nodes V', an *induced* subgraph of G is a subgraph where the edge set E' contains all edges from E that have both endpoints in V', formally $E' = \{(u, v) \in E | u, v \in V'\}$. Induced subgraphs are the most useful for this particular application as – physically speaking – all joints between elements in that appear in the parent structure should be included in any substructure.

Two graphs, G and H, share a *common* subgraph if both $G' \subseteq G$ *and* $G' \subseteq H$. Finding the largest common induced subgraph is a problem that has already been solved [9]. However, the Bron-Kerbosch algorithm described in Ref. [9] does not guarantee that the subgraphs are connected. If in fact, a subgraph consists of two disconnected components, it means it is representing two disconnected structures, and the two structures should be considered separately; therefore, for this particular application, only connected induced subgraphs will be considered.

Thankfully, a modification of the Bron-Kerbosch algorithm exists which ensures that only connected induced subgraphs are reported [10]. Since the Bron-Kerbosch algorithm only matches graphs based on their topology, the graphs are effectively treated as non-attributed. However, the attributes remain associated with the nodes, making it possible to access the attributes associated with any common subgraphs reported by the Bron-Kerbosch algorithm.

4.1.3. The modular product graph

The *modular product* is a graph (represented by $G \blacklozenge H$) produced by combining two graphs G and H. An example of a modular product graph is shown in Fig. 6. The modular product graph has the node set, generated by the Cartesian product $V(G) \times V(H)$ in which two nodes (x, u) and (y, v) are adjacent if:

- $(x, y) \in E(G)$ and $(u, v) \in E(H)$, or
- $(x, y) \notin E(G)$ and $(u, v) \notin E(H)$

These adjacency conditions for the modular product graph $G \blacklozenge H$ work like an XNOR gate, as shown in Table 5.

Now, a *clique* is a subset of nodes of an undirected graph such that every two distinct nodes in the clique are adjacent i.e. a *fully-connected* subgraph. A *maximal clique* in a graph is such that no other cliques have a greater number of nodes. An example of a maximal clique is shown in Fig. 7.

Examining Fig. 6 it can be seen that the cliques correspond to subgraphs which are common to both G and H. For example, the nodes A_1B_2 and A_2B_3 in the modular product correspond to the subgraphs G' and H', where $V(G') = \{A_1, A_2\} \subseteq V(G)$ and $V(H') = \{B_2, B_3\} \subseteq V(H)$, with corresponding edges $(A_1, A_2) \in E(G)$ and $(B_2, B_3) \in E(H)$. It is possible to verify that this is an induced subgraph. Larger cliques in the modular product graph correspond to larger common subgraphs. Maximum common subgraphs would correspond to maximal cliques.

However, induced subgraphs are not always connected. If an induced subgraph was formed from the nodes $A_1,\ B_1$ and $A_3,\ B_3$ instead, then $V(G') = \{A_1, A_3\}$. No edges would exist in this subgraph, since there are no edges in $E(G)$ with both endpoints in $V'(G)$, since the edge (A_1, A_3) is not in $E(G)$. The subgraph G' is still induced, but since no edges exist in G', it is disconnected. Likewise for the subgraph H', where $V(H') = \{B_1, B_3\}$.

While in this case, the nodes $A_1,\ B_1$ and $A_3,\ B_3$ do not represent a maximal clique and hence would not be reported, in more complicated modular product graphs, maximal cliques can correspond to induced subgraphs which are disconnected.

4.1.4. Solving the subgraph isomorphism problem

The maximum common induced subgraph problem is a generalisation of the subgraph isomorphism problem. The subgraph isomorphism problem involves taking two graphs G and H and determining whether or not G contains a subgraph that is isomorphic to H. The subgraph isomorphism problem can be solved by finding the maximal cliques in the modular product of the two graphs.

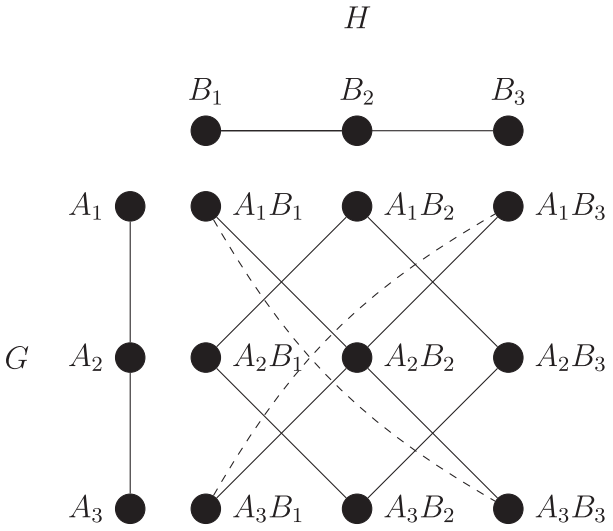

Fig. 6. The modular product graph for G and H. The solid lines represent c-edges and the dashed lines represent d-edges.

Table 5
Truth table showing the edges (adjacency) of the modular product graph $G \blacklozenge H$ based on the edges in the parent graphs G and H.

G	H	$G \blacklozenge H$
no edge	no edge	edge
no edge	edge	no edge
edge	no edge	no edge
edge	edge	edge

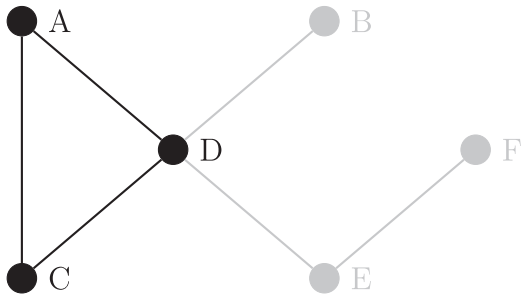

Fig. 7. A graph with a maximal clique highlighted.

The problem of finding all maximal cliques is NP-hard, since exhaustive search algorithms are required to solve it. This issue means that the computational time of any algorithm used to solve this problem will not scale for larger graphs; this also means that the problem of finding the maximum common induced subgraph between two graphs G and H is NP-complete.

4.2. Bron-Kerbosch algorithm

As mentioned previously, finding maximal cliques in the modular product of two graphs is equivalent to finding the common subgraphs. The Bron-Kerbosch algorithm [9] (described in Algorithm 1) will report all maximal cliques, and therefore all subgraphs between two graphs. This algorithm is guaranteed to generate induced subgraphs, but does not guarantee that they are connected. The Bron-Kerbosch algorithm is considered one the most efficient backtracking algorithms for maximal clique reporting and is widely used in chemical and life science applications [6]. The complexity of the Bron-Kerbosch algorithm is $\mathcal{O}(3^{n/3})$.

Algorithm 1. Bron-Kerbosch

R: set of nodes to be reported, initially $R = \varnothing$
P: set of nodes which can be added to R, initially $P = V$
X: set of nodes which cannot be added to R, initially $X = \varnothing$
1: **procedure** ReportCliques$R$, P, X
2: **if** $P = \varnothing$ **and** $X = \varnothing$ **then**
3: report R
4: **else**
5: Let P be the set $\{u_1, \ldots, u_k\}$
6: **for** $i \leftarrow 1$ **to** k **do**
7: $P \leftarrow P \setminus \{u_i\}$
8: $N \leftarrow \{v \in V | (u_i, v) \in E\}$
9: ReportCliques$R \cup \{u_i\}$, $P \cap N$, $X \cap N$
10: $X \leftarrow X \cup \{u_i\}$
11: **end for**
12: **end if**
13: **end procedure**

Algorithm 1 is intialised with R and X as empty sets (where R is the set of nodes to be reported, and X is the set of nodes which cannot be added to R). The algorithm is also initialised with the set $P = V$ (where P set of nodes that can be added to R).

The Bron-Kerbosch algorithm first checks whether both P and X are empty (line 2 in Algorithm 1). If P is empty, then there are no more nodes in the neighbourhood of the current node that can be added to the clique. If X is empty, then there are no nodes in the neighbourhood of the current node which have already been reported as part of a clique. Therefore, if both of these conditions are met, then a new maximal clique has been found. This maximal clique is then reported.

If this condition is not met, the Bron-Kerbosch algorithm iterates over the nodes in P. Since initially, $P = V$, the algorithm exhaustively searches the entire graph. The first step in the iteration is to remove the current node u_i from P. The next step in the iteration is to generate the neighbourhood of the current node N. Next, a recursive call of the algorithm is made, with the current node added to R, and P and X intersected with the neighbourhood of the current node. Using the intersection of P and X with N limits the next search to nodes adjacent to the current node (in the neighbourhood of) u_i. Once the subsequent recursive calls have been completed, the current node u_i is added to X.

The Bron-Kerbosch algorithm will report all of the maximal cliques in the graph. The maximal cliques in the graph shown in Fig. 8 will be $\{A, B, C\}$ and $\{C, D\}$. (A more detailed description of this specific example can be found in Appendix A.) The maximal cliques are then ranked in order of size to find the largest subgraph. Due to node re-labelling, any clique-finding algorithm will report all graphs that are isomorphic to a given subgraph.

Finding the subgraphs which give the best physical match is made possible through the use of attributes, as described in Section 5.

4.3. C-clique-finding algorithm

This algorithm is a modification of the Bron-Kerbosch algorithm which ensures that only connected induced subgraphs are found. This is not only useful for ensuring they are valid in a physical sense, but also significantly cuts down the search time. To limit the search to connected subgraphs, it is first necessary to define the concept of *c-edges*.

4.3.1. C-edges

Two nodes (x, u) and (y, v) in the modular product $G \blacklozenge H$ are adjacent via a c-edge (where the c stands for connected) if both $(x, y) \in E(G)$ and $(u, v) \in E(H)$. This is the first condition for adjacency in the modular product graph. The rest of the edges are defined by the second adjacency condition in the modular product graph and are known as d-edges (where the d stands for disconnected). These set of adjacency conditions are shown in Table 6. These adjacency conditions produce a modified modular product graph containing both c-edges and d-edges, as shown in Fig. 6.

Maximal cliques which include c-edges are called c-cliques. Limiting the search algorithm to only find c-cliques ensures that any reported subgraphs are connected. Only searching for c-cliques can also decrease the computation time [10] as the

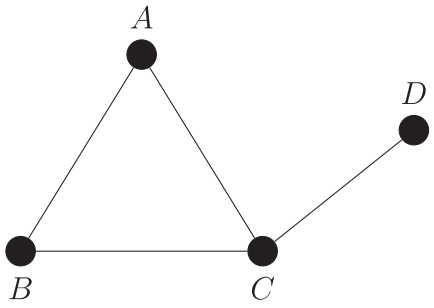

Fig. 8. A simple graph for illustrating clique-finding algorithms.

Table 6
Truth table showing the edges (adjacency) of the modular product graph $G \blacklozenge H$ based on the edges in the parent graphs G and H. If an edge exists in both parent graphs it is considered a c-edge. Where no edge exists in either of the parent graphs, it is considered a d-edge.

G	H	$G \blacklozenge H$
no edge	no edge	**d-edge**
no edge	edge	no edge
edge	no edge	no edge
edge	edge	**c-edge**

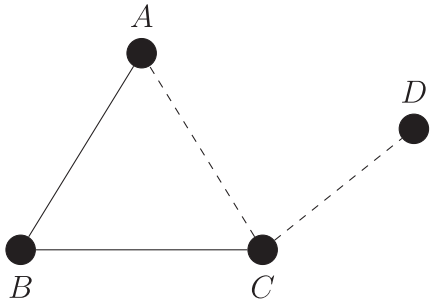

Fig. 9. A simple graph for illustrating clique-finding algorithms. The solid lines represent c-edges and the dashed lines represent d-edges.

set of c-cliques is a subset of all the maximal cliques in a graph. Therefore, a smaller number of cliques needs to be considered, which reduces the size of the search tree.

In Fig. 9, the only clique of interest is $\{A, B, C\}$ as the clique $\{C, D\}$ does not contain any c-edges.

To limit the search to only c-cliques, it is necessary to modify Algorithm 1. The major modification is the creation of a new set D which contains the set of nodes which cannot be directly added to the current clique because they are adjacent to the current node u via a d-edge.

This problem now requires an initialisation algorithm, shown in Algorithm 2. This initialisation algorithm iterates over all nodes in the graph, gradually adding them to T, which is the set of nodes that cannot be added to a clique because they have already been used to initialise the algorithm.

Algorithm 2. Initialisation for Report C-Cliques

```
R: set of nodes to be reported
P: set of nodes which can be added to R, because they are adjacent to node u via c-edges
D: set of nodes which cannot be directly added to R, because they are adjacent to u via d-edges
T: set of nodes which have already been used to initiate the REPORTC-CLIQUES algorithm
E_c: set of c-edges for the graph G
E_d: set of d-edges for the graph G
for u ∈ V do
    P ← ∅
    D ← ∅
    X ← ∅
    N ← {v ∈ V|(u, v) ∈ E}
    for v ∈ N do
        if (u, v) ∈ E_c then
            if v ∈ T then
                X ← X ∪ {v}
            else
                P ← P ∪ {v}
            end if
        else if (u, v) ∈ E_d then
            D ← D ∪ {v}
        end if
    end for
    R ← {u}
    ReportC-CliquesR, P, D, X, T
    T ← T ∪ {u}
end for
```

It is necessary to exclude nodes that have already been used to initialise the algorithm because otherwise every maximal clique of size n would be reported n times.

For the modified algorithm, R is the set of nodes to be reported; P is the set of nodes which can be added to R, because they are adjacent to node u via c-edges; D is the set of nodes which cannot be directly added to R, because they are adjacent to u via d-edges; and T is the set of nodes which have already been used to initiate the ReportC-Cliques algorithm.

The initialisation algorithm iterates through all nodes in the neighbourhood $N(v)$ and determines whether a node u is adjacent to v via a c-edge, in which case – provided that node has not been previously used to initialise ReportC-Cliques – the node is added to P. Nodes in P may be directly added to R in the next procedure call. If the node u has been previously used to initialise ReportC-Cliques, it is added to X instead. If the node u is adjacent to v via a d-edge, it is added to D. The node u is then added to R and the procedure ReportC-Cliques, described in Algorithm 3 is called. After the procedure has finished, u is added to T so that it is excluded from being added to future cliques. This procedure excludes cliques which do not contain a simple path of c-edges that connect all vertices within that clique (for example, $\{C, D\}$ in Fig. 9).

Algorithm 3. Modified Bron-Kerbosch

R: set of nodes to be reported
P: set of nodes which can be added to R, because they are adjacent to node u via c-edges
D: set of nodes which cannot be directly added to R, because they are adjacent to u via d-edges
T: set of nodes which have already been used to initiate the ReportC-Cliques algorithm
E_c: set of c-edges for the graph G

```
1: procedure ReportC-CliquesR, P, D, X, T
2:   if P = ∅ and X = ∅ then
3:     report R
4:   else
5:     Let P be the set {u_1, ..., u_k}
6:     for i ← 1 to k do
7:       P ← P \ {u_i}
8:       N ← {v ∈ V|(u_i, v) ∈ E}
9:       for all v ∈ D do
10:        if (v, u_i) ∈ E_c then
11:          if v ∈ T then
12:            X ← X ∪ {v}
13:          else
14:            P ← P ∪ {v}
15:          end if
16:          D ← D \ {v}
17:        end if
18:      end for
19:      ReportC-CliquesR ∪ {u_i}, P ∩ N, D ∩ N, X ∩ N, T
20:      X ← X ∪ {u_i}
21:    end for
22:  end if
23: end procedure
```

A node in D cannot be added to R at the point where ReportC-Cliques is called. However, nodes in D must be checked as they may at some future point become adjacent to a node in P via a c-edge. This check is performed on Lines 9 and 10 in Algorithm 3, where each node in D is checked for adjacency via a c-edge with the current node u_i. If the node u_i is adjacent to v via a c-edge, and has not previously been used to initialise the procedure ReportC-Cliques (i.e. $v \notin T$), then the node is added to P and can potentially be added to R in the next procedure call. If the node $v \in T$ then v is added to X. The node v is then removed from D. Aside from this check to see whether or not nodes can be moved from D to P, Algorithm 3 is essentially the same as Algorithm 1.

For the remainder of the discussion, all common subgraphs are assumed to be connected and induced.

5. Networks and communities

The six structures that form the network that will be used for illustration here are representative of the type of structures typically encountered in structural health monitoring problems: t1 represents an onshore wind turbine; a1 represents an aeroplane similar to a Boeing-747; a2 represents an aeroplane similar to a Cessna 172; b1 represents a bridge with a single support column, similar to a motorway bridge; b2 represents a bridge with two support columns; and b3 represents a bridge with three support columns. The AGs corresponding to the structures can be seen on the diagonal of Fig. 10.

The set of common subgraphs found by Algorithm 3 was reduced to subgraphs where the set of boundary nodes V_{BC} for each graph were aligned. Since boundary nodes do not represent a physical part of the structure, it does not make sense to compare them with other node types. Therefore, a common subgraph was excluded if for *any* node (u, v) in the common subgraph:

- $u \in V_{BC}(G)$ and $v \notin V_{BC}(H)$, or
- $u \notin V_{BC}(G)$ and $v \in V_{BC}(G)$

The maximum common subgraph for each pair of structures where the boundary conditions are aligned as shown in Fig. 10.

Once this reduced set of common subgraphs has been found, it is possible to determine a similarity score. Here the similarity score used is the *Jaccard similarity coefficient* [14]. While there are other similarity metrics available, the Jaccard sim-

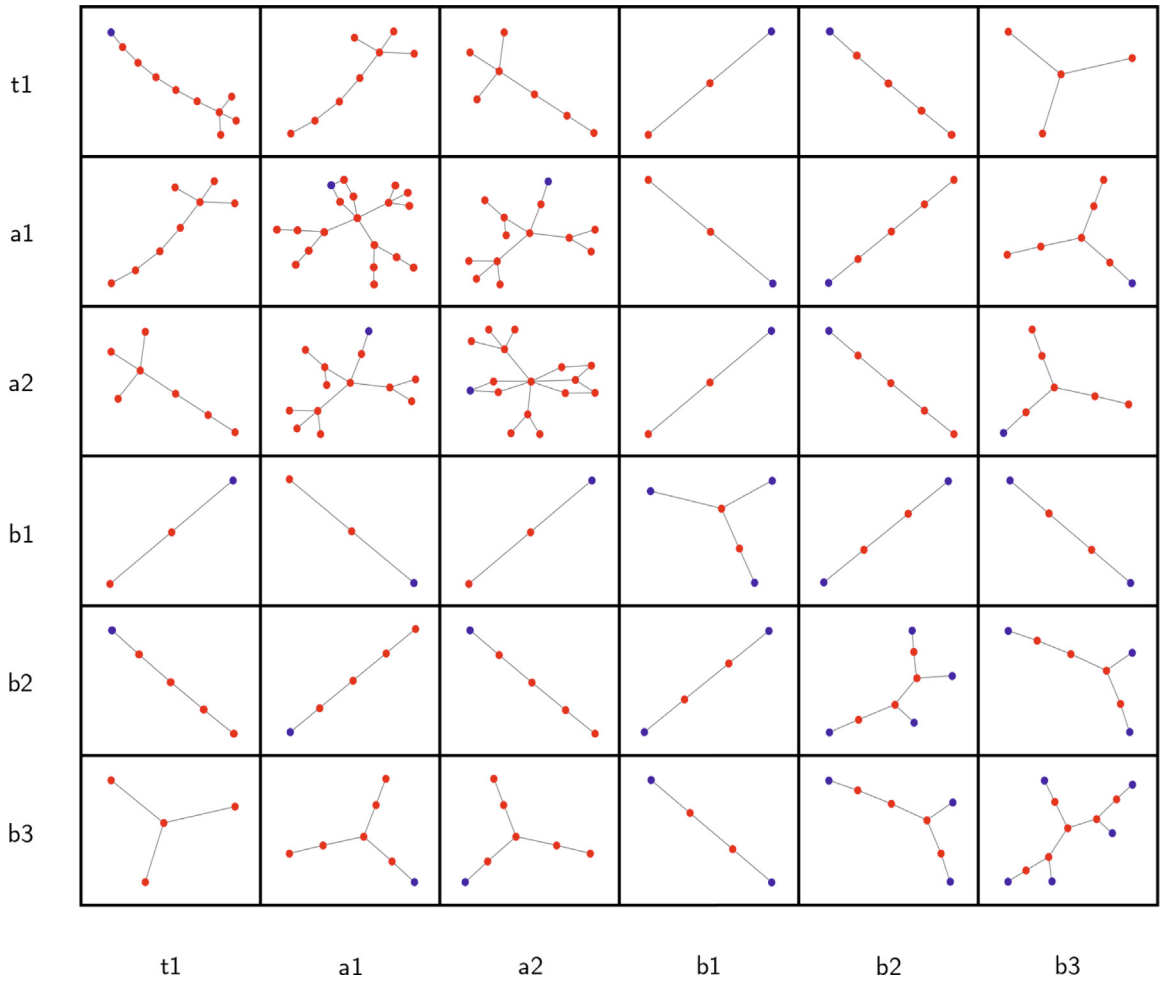

Fig. 10. The maximum common subgraphs (with aligned boundary nodes) for each pair of structures. The boundary nodes are coloured blue. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

ilarity coefficient is commonly used and sufficient for the purposes of this paper. The Jaccard similarity coefficient is based on the size of the maximum common subgraph G'; thus providing a measure of the similarity based purely on the topology of the structures. The Jaccard similarity coefficient is calculated using the following equation,

$$J_{\mathrm{v}}(G,H) = \frac{|V(G')|}{|V(G)| + |V(H)| - |V(G')|} \tag{1}$$

where $J_{\mathrm{v}}(G,H)$ is the Jaccard index for the node sets.

The Jaccard index can be used to generate the Jaccard distance $D_{\mathrm{v}}(G,H) = 1 - J_{\mathrm{v}}(G,H)$. The Jaccard distance is a metric on the set of structures. The resulting Jaccard distances for a set of six structures are shown in Fig. 11. The resulting distance matrix is the *network* of structures. The network can also be represented graphically, as shown in Fig. 12.

By grouping structures according to their distances, it is possible to create *communities* of structures. Within these communities, it is expected that a certain level of knowledge transfer is possible. The communities which are formed may change based on the particular SHM problem, depending on which attributes are determined to be the most relevant. Creating communities in the network shown in Fig. 11 is achieved through the use of clustering algorithms.

Applying a simple k-nearest neighbours algorithm to Fig. 11, with $k = 1$ gives the following results: t1 is equidistant to a1 and b2, and so would form its own class; the structures a1 and a2 are closest to one another, forming a community of aeroplanes; and the bridge b1 is closest to b2, and b2 is closest to b3, thereby forming a community of bridges.

6. Attribute matching to determine homogeneity

Further similarity scores can be calculated by examining the node attributes in each graph that form the subgraph in each question. The subgraph node labels contain the node labels from each parent graph. These node labels can the be used to

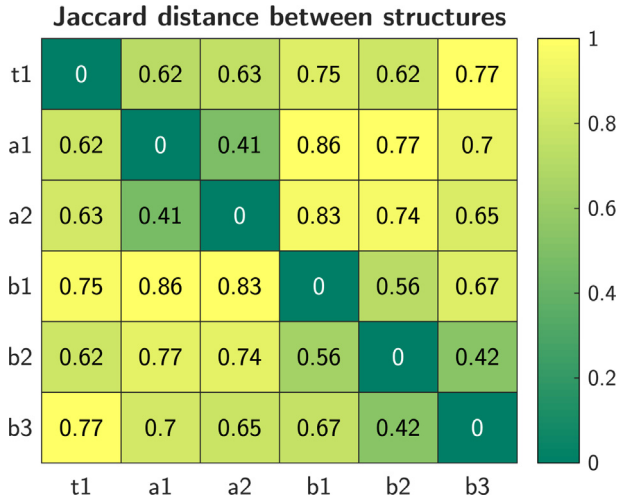

Fig. 11. Figures showing the Jaccard distance between structures in the network. A distance of 0 indicates that structures are identical, whereas a distance of 1 indicates that structures have nothing in common.

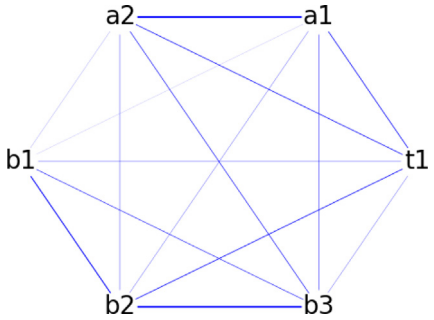

Fig. 12. A graphical representation of the network, based on the distances in Fig. 11. The thickness of the edges indicates that structures are more similar.

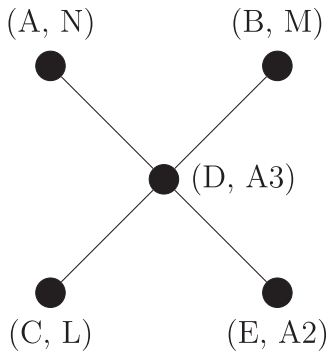

Fig. 13. A possible subgraph from the two graphs shown in Fig. 4. The left-hand entries in the node labels correspond to the wind turbine, while the right-hand entries correspond to the aeroplane.

query the dictionary containing the node attributes for each parent graph. For the sample subgraph shown in Fig. 13, the node labels can be used to look up the attributes for the corresponding nodes in the original AGs, shown in Fig. 4.

The left-hand entries in node labels (A, B, C, D, E) correspond to the wind turbine, and so node and edge attributes can be found in Tables 1 and 3, whereas the right-hand entries in the node labels (A2, A3, N, M, L) correspond to the aeroplane and can be used to find node attributes from Tables 2 and 4. For example, the geometry class and shape for node in the wind turbine is 'Beam, Aerofoil', and for node N in the aeroplane the geometry class and shape is also 'Beam, Aerofoil' so there is a match. However, node D in the wind turbine represents *complex* geometry, while node A3 in the aeroplane represents a *shell*, so these do not match.

The comparison is performed at the greatest possible level of resolution. For example, when comparing the geometry attributes, if the geometry class matches, then the shape is compared. If the shape matches, then the dimensions for each are examined. If the elements match on all levels, then they are essentially identical elements. This is shown as a flowchart in Fig. 14.

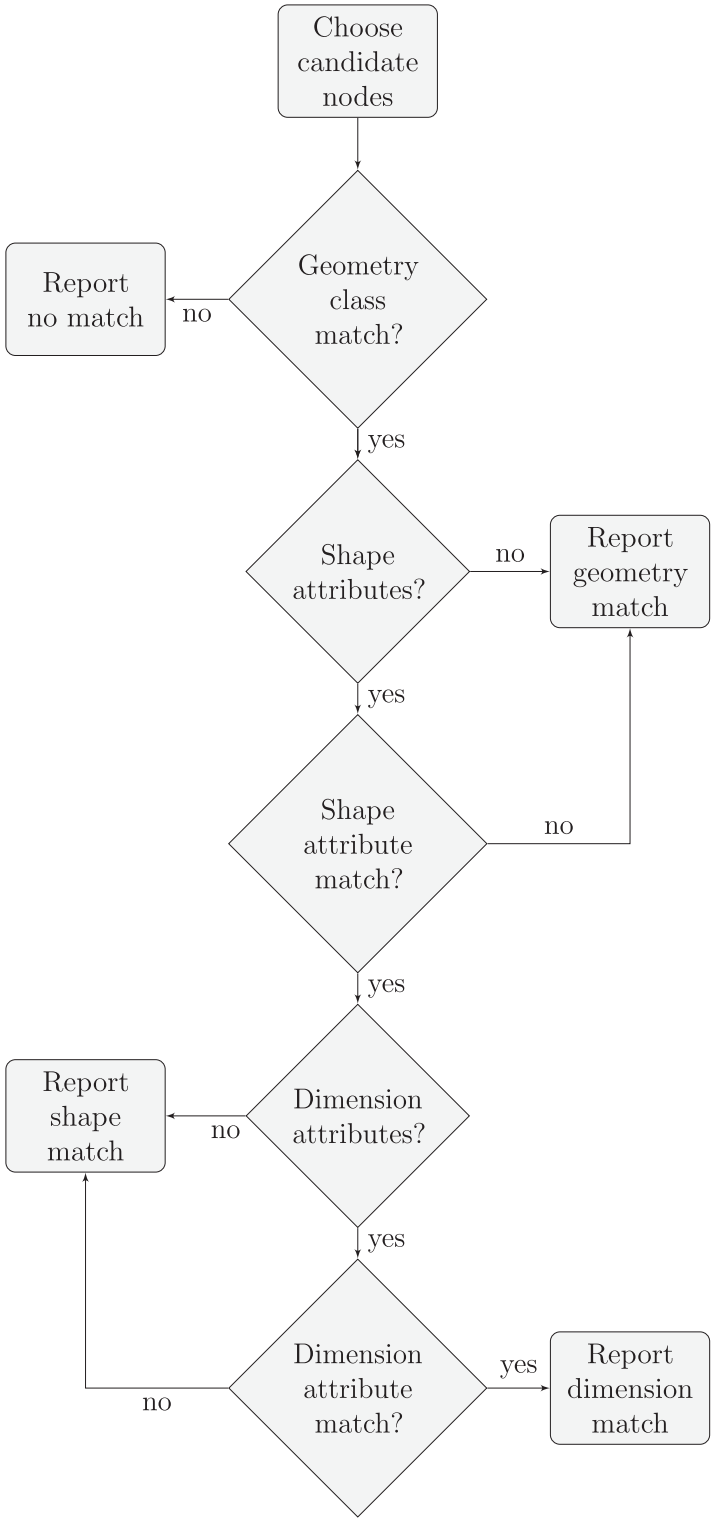

Fig. 14. A flowchart for comparison of the geometry attributes of a pair of nodes in the AG.

This is true for the case of materials as well. If the material class, specific material and material properties are all the same, then the two elements are made from the same material.

The same matching can be performed for joints. The joint to examine in each parent graph is determined by examining the edges in the subgraph. Since the subgraphs are induced, each edge in the subgraph corresponds to an edge in each parent graph. From this, the joint attributes can be extracted. The joint type is compared, and if the joint is kinematic, the restricted degrees of freedom can also be compared.

If all elements and joints in two graphs are identical (all of the hierarchical attributes are available and match) then these two graphs form a homogeneous population. If all elements and joints in the subgraph of two graphs are identical, then they share a subcomponent and transfer is possible within this subcomponent [11]. The more likely case is that only partial matches are seen, and in this case the level of match determines the appropriate transfer learning approach.

7. Some formal definitions for PBSHM

The details of the IE and AG constructions in this paper will now allow formal definitions for some of the terms that have been used up to now, like *homogeneous* and *heterogeneous* populations. The various terms will prove important when considering the problems of transfer [11].

First of all a *population* of structures will simply be some set of N structures, $\{S_1, \ldots, S_N\}$, which exist in reality. With each structure S_i, one can associate/construct an appropriate IE model $IE(S_i)$ or IE_i. Each of the IE models will induce an attributed graph representation $AG(S_i)$ or AG_i.

Two structures S_1 and S_2 will be said to be *topologically equivalent* if they are topologically equivalent, when considered as AGs, i.e. there are the same number of nodes in each, and the pattern of connectivity is the same. Topological equivalency necessarily implies that the corresponding graphs, G_1 and G_2, are isomorphic.

If two graphs G_1 and G_2 are topologically equivalent, and the boundary conditions in both AGs occur in corresponding places (as in Fig. 10), then the graphs are *structurally equivalent*. Since the AGs are induced by the IE representations, structural equivalency of the graphs implies that S_1 and S_2 are structurally equivalent.

Two structures will be said to be *strongly equivalent* if they are structurally equivalent, but also have corresponding IEs at corresponding nodes in the AG; this means that the attribute vectors for the nodes must have the same entries, with the same physical meanings.

Consider a population $P = \{S_i : i = 1, \ldots, N\}$, where each structure S_i has parameters representing the node attributes, denoted by $\underline{\theta}_j^i$, where j is the node index, and parameters $\underline{\theta}_{jk}^i$, where j and k indicate that the edge is between nodes j and k. If all pairs of structures are strongly equivalent, the parameters in the vectors for each structure are in one-to-one correspondence in terms of physical meaning. A population now, will be termed *homogeneous* if the attributes can be considered to be random draws from distributions $p_j(\underline{\theta}_j)$ and $p_{jk}(\underline{\theta}_{jk})$, which are *independent* of the structure. Clearly, this definition encompasses the idea that a homogeneous population is comprised of structures that are exactly the same make and model, with variations arising only from manufacturing/embodiment variations.

In order to fully motivate the idea of a *form* [1], it may be necessary to add a stronger condition, even than this. For example, one might say that a population is *strongly homogeneous* if it is homogeneous, and the attribute densities p_j and p_{jk} are unimodal, or convex, with low dispersion. For example, a wind farm with all the turbines the same model, but subject to manufacturing variations would be expected to be strongly homogeneous, assuming that some parameterised foundation model was appropriate to all the turbines.

If a population is not *homogeneous* in terms of pairwise correspondence between structures, it is termed *heterogeneous*.

These definitions are part of the essential machinery that will be needed to assess if transfer of inferences is likely to succeed between two different structures [11].

8. Alternative methods of graph comparison

The methods explored in this paper focus on using a modified BK algorithm to find the MCS between two graphs and then using the size of the MCS as a measure of the similarity. Of course, the BK algorithm is just one possible way to find the MCS. The BK algorithm exploits the modular product graph (also known as the association graph), whereas other methods employ backtracking, where a search tree containing all possible vertex correspondences is constructed instead. This search tree can be explored using a depth-first search [15] or breadth-first search [16]. These papers come from a bioinformatics approach to molecular biology, using graphs to assess the similarity of molecules and gene transcription factors. There are also other modifications that limit the search to vertex correspondences where the vertices have matching attributes such as in Ref. [17]. For a comprehensive review of MCS-based approaches to determining graph similarity with a focus on biology and chemistry, the reader is referred to Ref. [6].

While the MCS-based approach may be computationally expensive, it has the benefit of returning a list of the subgraphs that are common to two graphs, which depending on the context, may represent either significant groups in a molecule or an important signalling pathway within a cell. In fact, subgraphs can hold such significance that searching for frequently-occurring subgraphs in the interaction networks of cells is a technique for discovering the specific biological functions that form the building blocks of these networks [18].

Of course, these subgraph-based techniques come with a high computational cost. A different approach to analysing graphs (in particular networks), is to examine the properties of the graph itself, for example the modularity of a network can be used to divide the network into communities [19]. The spectra of the Laplacian matrix for two different graphs can be used as a measure of their topological similarity [20]. A brief introduction to spectral graph theory can be found in Ref. [21]. While these approaches are less computationally expensive, they cannot provide specific information as to the location of similarities or differences within graphs.

Recently, with the rise of neural networks and an increased interest in graph-structured data, machine learning has been used to analyse graphs. There are too many architectures to describe here, but fortunately a large number are described in [22], which is a broad introduction to the use of deep learning techniques in graph-based analysis. Instead of going into each method and application in detail, there will be a few examples provided to give a flavour of the techniques and applications that exist. Graph Neural Networks have been applied to security by examining the similarity of various functions within a piece of software [23]. Graph Neural Networks have also been used to predict out-of-knowledge-base entities for Knowledge Graphs [24]. Gated Graph Neural Networks have been applied to natural language processing for improved machine translation [25], while previous approaches relied on Recurrent Neural Networks [26]. Convolutional neural networks have been applied directly on graphs to learn molecular fingerprints [27]. Neural networks have even been used to generate new graphs, which could be relevant to drug discovery by generating the graphs for new molecules with specific properties [28].

9. Conclusions

Population-based structural health monitoring relies on the idea that it is possible to transfer knowledge between structures. For structures which form a homogeneous population, models (such as a form [1]) and data can be shared directly. However, for structures which form a heterogeneous population, it is necessary to somehow establish the structural similarity to determine whether knowledge transfer is possible.

To compare the similarity of two structures, they are first converted into Irreducible Element (IE) models, which abstract features from the structures that are important for knowledge transfer. These features are captured in the properties of the IE models. The degree of similarity between the properties of the IE models for two structures determines the level of inference possible. Matching properties ensures the consistent labels required for transfer learning. Matching topology and element geometry gives consistent location labels. Matching the material properties for elements gives consistent damage classification and assessment labels. Therefore, it is important to consider all three aspects of a particular structure (topology, materials and geometry) when determining whether knowledge transfer is possible [11].

IE models are made up of elements connected by joints. The elements in an IE model capture the material and geometrical properties. The joints carry important topological information. Together the elements and joints capture the features that will significantly affect knowledge transfer.

Once the IE model for a structure has been generated, it is then converted into an attributed graph (AG). Accordingly, the AGs capture information relating to the three categories (topology, materials, and geometry). The nodes of the AG (which correspond to the elements of the IE) contain attributes which correspond to these properties in the IE, however the information is arranged in a way that makes it more accessible to any graph-matching algorithms. The edges of the AG (which correspond to joints in the IE) carry important topological information, particularly in terms of connections to ground, which is necessary for not only creating the AG, but also when searching for common subgraphs.

The method for finding common subgraphs from two parent graphs is to first find the modular product of the two graphs, and then use a clique-finding algorithm to generate a list of subgraphs. The nodes in the subgraphs correspond to a set of nodes in each parent graph. The attributes of the corresponding nodes in each parent graph can then be compared. Comparing node attributes is simple, as the nodes in each graph are uniquely identified (within a particular graph) and the information can be stored using a variety of data-structures, including databases. Transferring this methodology to a database will form the basis of future work.

By carefully choosing the information and data structure, it is possible to compare the AG for two structures in order to determine the most appropriate transfer learning tool. Comparing the attributes of the graphs (which correspond to the properties of the IE model) allows one to group structures based on the level of knowledge transfer possible between any two given structures. These issues will be discussed in detail in the final paper in this sequence [11].

CRediT authorship contribution statement

J. Gosliga: Methodology, Conceptualization, Formal analysis, Investigation, Writing - review & editing. **P.A. Gardner:** Methodology, Conceptualization, Writing - review & editing. **L.A. Bull:** Methodology, Writing - review & editing. **N. Dervilis:** Methodology, Writing - review & editing, Supervision. **K. Worden:** Conceptualization, Methodology, Funding acquisition, Writing - review & editing, Supervision.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors would like to thank the UK EPSRC for funding through the Established Career Fellowship EP/R003645/1 and the Programme Grant EP/R006768/1.

Appendix A. Bron-Kerbosch example

A simple example using the graph shown in Fig. 8 will be used to illustrate how the algorithm works, although the recursive nature of the algorithm makes it hard to describe, so it is recommended to work through the example, writing out the sets at each step.

Recursive level 1: when the procedure ReportC-Cliques is first called $P \neq \varnothing$. The procedure then iterate through the nodes in P, which in this example makes $u_1 = A$. The node u_1 is then removed from the set nodes P which can be added to the clique. The open neighbourhood $N(A)$ is generated. In this case, $N = \{B, C\}$. The algorithm is then called recursively with A now added to the current clique R. The set of nodes of which can be added to the clique is now limited to nodes within the neighbourhood of A. The set of nodes which are excluded from being added is also now limited to nodes within the neighbourhood of A.

Recursive level 2: for the second recursive call, $P = \{B, C\} \neq \varnothing$, and so the next node B is chosen. This is now removed from P and the neighbourhood is generated. The next recursive call is made, where B is added to $R, P = \{C\}$ only, and $X = \varnothing$.

Recursive level 3: for the third recursive call, $P = C$ initially, so the algorithm continues. C is removed from P, the neighbourhood $N(C)$ is generated. In the next call, C is added to R, but $P \cap N$ and $X \cap N$ are now empty, so when Line 3 is reached, the clique $R = \{A, B, C\}$ is reported, and the procedure terminates. This returns to the previous level of recursion as C is added to X. Since there are no more nodes in P, the procedure terminates.

Recursive level 2: the algorithm returns to the previous level where $P = \{B, C\}$ originally and $R = \{A, B\}$. B is now added to X and $u_2 = C$.

Recursive level 3: the algorithm is called with $P \cap N(C) = \varnothing, X \cap N(C) = \{B\}$. Even though there are no more nodes in P to add, the clique $R = \{A, B, C\}$ is not reported again since X is not empty. Also, since P is empty, there are no nodes to loop over, and so the procedure is terminated and the algorithm returns to the previous recursive level.

Recursive level 2: C is now added to X and the procedure terminates and returns to the previous level.

Recursive level 1: A is added to X and now $u_2 = B$. Now $P = \{C, D\}$.

Recursive level 2: in this call $P \cap N(B) = \{C\}, X \cap N(B) = \{A\}, R = \{A, B\}$. No clique is reported, and $u_1 = C$. C is then removed from P and ReportC-Cliques is called again.

Recursive level 3: in this call $P \cap N(C) = \varnothing$ and $R = \{A, B, C\}$, but since $X \cap N(C) = \{A\}$, the clique R is not reported. Since P is empty, the loop cannot be performed and the procedure terminates.

Recursive level 2: C is now moved to X. Since C was the only member of P, the loop has now finished and the procedure terminates.

Recursive level 1: now $u_3 = C, P = \{D\}$ and $X = \{A, B\}$.

Recursive level 2: in this call $R = \{C\}, P \cap N(C) = \{D\}$ and $X \cap N(C) = \{A, B\}$. $u_1 = D$ and D is removed from P.

Recursive level 3: in this call $P \cap N(D) = \varnothing$ and $X \cap N(D) = \varnothing$ so the clique $R = \{C, D\}$ is reported. The procedure then terminates.

Recursive level 2: D is added to X and since P is empty, the loop ends and the procedure terminates.

Recursive level 1: now $X = \{A, B, C\}$ which prevents the clique $R = \{C, D\}$ being reported again when $u_4 = D$. From visual inspection of Fig. 8 it is possible to determine that no more maximal cliques are present.

References

[1] L.A. Bull, P.A. Gardner, J. Gosliga, T.J. Rogers, N. Dervilis, E.J. Cross, E. Papatheou, A.E. Maguire, C. Campos, K. Worden, Foundations of population-based SHM, Part I: homogeneous populations and forms. Submitted to, Mechanical Systems and Signal Processing (2020).
[2] C.R. Farrar, K. Worden, Structural Health Monitoring: A Machine Learning Perspective, John Wiley and Sons, 2012.
[3] A. Rytter, Vibrational based inspection of civil engineering structures. PhD thesis, Deptartment of Building Technology and Structural Engineering, Aalborg University, Denmark, 1993.
[4] I. Antoniadou, N. Dervilis, E. Papatheou, A.E. Maguire, K. Worden, Aspects of structural health and condition monitoring of offshore wind turbines, Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 373 (2015).
[5] E. Papatheou, N. Dervilis, A.E. Maguire, I. Antoniadou, K. Worden, A performance monitoring approach for the novel Lillgrund offshore wind farm, IEEE Transactions on Industrial Electronics 62 (2015) 6636–6644.
[6] E. Duesbury, J.D. Holliday, P. Willett, Maximum common subgraph isomorphism algorithms, MATCH Communications in Mathematical and in Computer Chemistry 77 (2017) 213–232.
[7] M. El-Mehalawi, R. Ällen Miller, A database system of mechanical components based on geometric and topological similarity. Part I: Representation, Computer Aided Design 35 (2001) 83–94.

[8] M. El-Mehalawi, R.Ãllen Miller. A database system of mechanical components based on geometric and topological similarity. Part II: Indexing, retrieval, matching, and similarity assessment, Computer Aided Design 35 (2001).
[9] C. Bron, J. Kerbosch, Algorithm 457: finding all cliques of an undirected graph, Communications of the ACM 16 (1973) 575–577.
[10] I. Koch, Enumerating all connected maximal common subgraphs in two graphs, Theoretical Computer Science 250 (2001) 1–30.
[11] P.A. Gardner, L.A. Bull, J. Gosliga, N. Dervilis, K. Worden, Foundations of population-based SHM, Part III: heterogeneous populations – transfer and mapping, Submitted to Mechanical Systems and Signal Processing, 2020.
[12] P. Gardner, X. Liu, K. Worden, On the application of domain adaptation in structural health monitoring, Mechanical Systems and Signal Processing 138 (2019), 106550.
[13] R. Diestel, Graph Theory, third ed., Springer-Verlag, New York, 2006.
[14] P. Jaccard, Étude comparative de la distribution florale dans une portion des alpes et des jura, Bulletin de la Société vaudoise des sciences naturelles 37 (1901) 547–579.
[15] Y. Cao, T. Jiang, T. Girke, A maximum common substructure-based algorithm for searching and predicting drug-like compounds, Intelligent Systems for Molecular Biology 24 (2008).
[16] R.J.P. van Berlo, W. Winterbach, M.J.L. de Groot, A. Bender, P.J.T. Verheijen, M.J.T. Reinders, D. de Ridder, Efficient calculation of compound similarity based on maximum common subgraphs and its application to prediction of gene transcript levels, International Journal of Bioinformatics Research and Applications 9 (2013) 407–432.
[17] J.W. Raymond, E.J. Gardiner, P. Willett, Rascal: Calculation of graph similarity using maximum common edge subgraphs, The Computer Journal 45 (2002).
[18] A. Mrzic, P. Meysman, W. Bittermieux, P. Moris, B. Cule, B. Goethals, K. Laukens, Grasping frequent subgraph mining for bioinformatics applications, BioData Mining (2018).
[19] M. Newman, Networks, Oxford University Press, 2018.
[20] Y. Shimada, Y. Hirata, T. Ikeguchi, K. Aihara, Graph distance for complex networks, Scientific Reports (2016).
[21] D.A. Spielman, Spectral graph theory and its applications, in: 48th Annual IEEE Symposium on Foundations of Computer Science, 2007.
[22] D. Bacciu, F. Errica, A. Micheli, M. Podda, A gentle introduction to deep learning for graphs, Neural Networks (2020).
[23] Y. Li, C. Gu, T. Dullien, O. Vinyals, P. Kohli, Graph matching networks for learning the similarity of graph structured objects, in: Proceedings of the 36th International Conference on Machine Learning, 2019.
[24] T. Hamaguchi, H. Oiwa, M. Shimbo, Y. Matsumoto, Knowledge transfer for out-of-knowledge-base entities: a graph neural network approach, in: Proceedings of the 26th International Joint Conference on Artificial Intelligence, 2017.
[25] D. Beck, G. Haffari, T. Cohn, Graph-to-sequence learning using gated graph neural networks, 2018.
[26] I. Sutskever, O. Vinyals, Q.V. Le, Sequence to sequence learning with neural networks, in: Proceedings of the 27th International Conference on Neural Information Processing Systems, vol. 2.
[27] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, Timothy Hirzel, A. Aspuru-Guzik, R.P. Adams, Convolutional networks on graphs for learning molecular fingerprints, in: Proceedings of Advances in Neural Information Processing Systems, vol. 28, 2015.
[28] R. Liao, Y. Li, Y. Song, S. Wang, C. Nash, W.L. Hamilton, D. Duvenaud, R. Urtasun, R. Zemel, Efficient graph generation with graph recurrent attention networks, in: 33rd Conference on Neural Information Processing Systems, 2019.