Graph Extraction for Assisting Crash Simulation Data Analysis

Anahita Pakiman[1,2], Jochen Garcke[1,3], and Axel Schumacher[2]

[1] Fraunhofer SCAI, Germany
anahita.pakiman@scai.fraunhofer.de
[2] Bergische Universität Wuppertal, Germany
[3] Institut für Numerische Simulation, Universität Bonn, Germany

Abstract. In this work, we establish a method for abstracting information from Computer Aided Engineering (CAE) into graphs. Such graph representations of CAE data can improve design guidelines and support recommendation systems by enabling the comparison of simulations, highlighting unexplored experimental designs, and correlating different designs. We focus on the load-path in crashworthiness analysis, a complex sub-discipline in vehicle design. The load-path is the sequence of parts that absorb most of the energy caused by the impact. To detect the load-path, we generate a directed weighted graph from the CAE data. The vertices represent the vehicle's parts, and the edges are an abstraction of the connectivity of the parts. The edge direction follows the temporal occurrence of the collision, where the edge weights reflect aspects of the energy absorption. We introduce and assess three methods for graph extraction and an additional method for further updating each graph with the sequences of absorption. Based on longest-path calculations, we introduce an automated detection of the load-path, which we analyse for the different graph extraction methods and weights. Finally, we show how our method for the detection of load-paths helps in the classification and labelling of CAE simulations.

Keywords: Automotive · CAE Knowledge · Graph Extraction· Weighted-Directed Graph · Flow Calculation · Load-path Detection

1 Introduction

We live in an interconnected world, and graph theory provides powerful tools for modelling and analysing this interconnectedness. In graph theory, graphs are usually given in advance or easily abstracted from problems. However, for many real-world scenarios, the individual data instantiations of modelled graphs need to be determined from the data before further analysis. Therefore, the construction of high-quality graphs has become an increasingly desirable research problem, resulting in many graph construction methods in recent years [1]. Furthermore, knowledge graph (KG)s have become a new form of knowledge representation and are the cornerstone of several applications for specific use cases in industry. The graph underlying the abstract structure, which effectively facilitates domain conceptualisation and data management, is the reason for the growing interest in this technology. Moreover, the use of KG is the direct driver of several artificial intelligence applications [2]. Towards vehicle KG, we aim to capture knowledge about vehicle development designs by automatically extracting graphs from a finite element (FE) model representing a vehicle.

The simplest scenario for identifying the connectivity of a graph is when it is associated with a physical problem related to the graph. Such graphs include electrical circuits, power grids, linear heat transfer, social and computer networks, and spring-mass systems [3]. In this work, we are interested in crashworthiness studies in vehicle design, where the transformation of crash simulation data into a graph is a challenging and unexplored area of research. With the resulting representation, we aim to provide an abstraction of the problem that allows the use of graph theory methods for further automated analysis of the simulations.

Computer aided engineering (CAE) analysis, mostly with the finite element method (FEM), enables car manufacturers to analyse many design scenarios, nowadays between 10,000 to 30,000 simulations per week [4]. In crashworthiness analysis, CAE engineers optimise the distribution of impact energy in the vehicle structure to reduce injuries to occupants or vulnerable road users. How to characterise the sequence of absorbed energy, known as the load-path, is a fundamental question in this analysis. The results of crash simulations include several outputs, such as deformations, accelerations and internal energy. However, the load-path is not explicitly calculated in a crash simulation. Therefore, a CAE engineer must visualise the sequence to reveal the load-path. In this work, we propose and investigate graph representations for an automated identification of the load-path from the simulation data.

We consider parts of the FE model entities as vertices of the structural graph following the scheme of [5]. We want to detect the graph edges that resemble the structural connectivity of the vehicle. We propose three approaches to determine this structural graph: component-based graph (CBG), single part-based graph (sPBG) and multi part-based graph (mPBG). The CBG follows two steps: finding the connection of the components (a group of parts) and then identifying the connection of the parts in each component. The sPBG and mPBG graphs have additional steps to convert the component connections to part connections, which requires the detection of the parts that are entangled in the connection that is supporting the flow of energy.

Defining the vehicle structure as a graph is the first step in load-path detection. Secondly, we compute it as the longest path in weighted directed graphs, where the edge weights between the parts shall represent the energy flow during the crash. We study different edge weighting functions for three graph extraction scenarios and analyse the determined load-paths from an engineering perspective. In this work, the investigation is carried out on the frontal structure of a complete vehicle with a multi-scenario load-path in a full frontal load case. But, our approach is applicable to different impact directions and load case scenarios.

In summary, the main contributions of this work are:

– the conversion of a vehicle structure to a weighted directed graph,
– the extraction of features representing the energy flow,
– a further graph segmentation that captures the time sequence of events,
– an automated detection of the load-path,
– the clustering of simulations based on their load-paths.

2 Related work

Recently, a graph schema to model vehicle development with a focus on crash safety was introduced in [5]. The graph modelling considers the CAE data in the context of the R&D development process and vehicle safety, with the aim to enable searchability, filtering, recommendation, and prediction for crash CAE data during the development process. In [5], the car parts are directly connected to their simulation, and the parts between the simulations have a connection to similar design based on the properties ID (PID) of the parts. But, connections between the parts of one simulation are missing, therefore the vehicle's structure and its connectivity is not modelled. Thus, incorporating the vehicle structure into the graph structure will enrich the data representation.

In crashworthiness, graphs have been used to predict the response of the vehicle [6] or barrier [7] with so-called bond graphs. The bond graphs available for vehicle crashes represent the problem from the perspective of a mass-spring model [6]. Bond graphs are ideal for visualising the essential properties of a system because their graphical nature separates the system structure from the equations. [8]. Bond graphs represent the vehicle structure by summarising the physical elements and connections. However, to the best of our knowledge, there is no way of automatically extracting the vehicle structure as a bond graph.

Before the growth of computing power allowed large FEM analysis, there were other modelling techniques that simplified the problem to a mass-spring model. The advantage of the mass-spring model is that it can be easily represented as a weighted graph. SISAME (Structural Impact Simulation And Model-Extraction) is a general-purpose tool for the extraction and simulation of one-dimensional non-linear lumped parameter structural models [9]. Using SISAME, mass element weights and spring element load-paths were optimally extracted directly from the test data accelerations and wall forces [10]. However, the lumped mass spring (LMS) modelling is one-dimensional and focuses mainly on accurately modelling the test data rather than representing the structural performance of the vehicle. Later the deformation space models (DSM) model was introduced [11] to compensate for the limitations of the LMS. It can only roughly capture displacements and energy absorption, neglecting connections and interactions with other components.

Another use of graphs in crash analysis is in the structural optimisation of the vehicle [12, 13]. Here, the optimisation method adds vertices and edges to stiffen the structure, starting with a simple graph describing the perimeter of the vehicle. The focus of these studies is to search with a graph for the optimal solution of the vehicle design. As a result, to complete the vehicle design and ensure safety performance, further processes and CAE analysis are required.

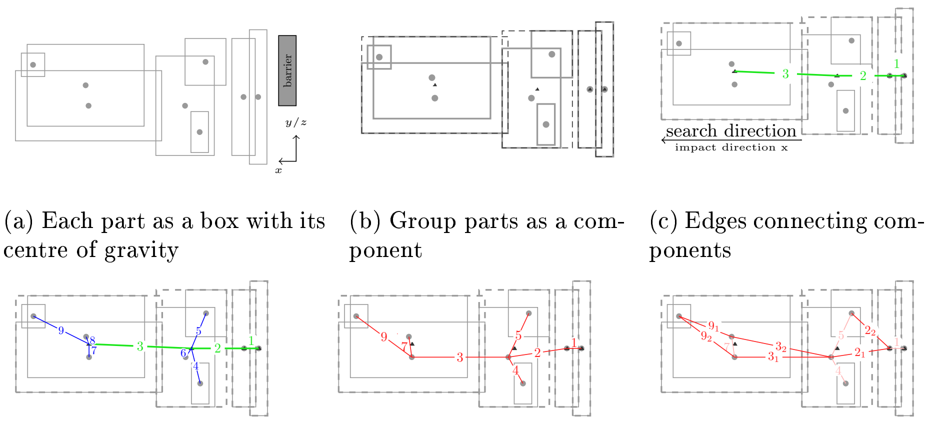
To summarise, automatically converting a crash FE model in vehicle development to a graph is still an open research question. Depending on the detail required in a graph, there are several ways to represent an FE model of a vehicle. As a specific application, we investigate how adding connections to the graph will allow a load-path analysis for each simulation. For that, we use and extend the recently introduced energy absorption features [14], which characterize the simulation's behaviour, as edge features to enable the load-path detection.

3 Graph extraction

It is a challenging task to generate a graph representing the structure of a vehicle from CAE data. Finding the connectivity of the parts is complex due to the number of connections, the variety of FE modelling techniques and the variety of physical types of connections. The best way to obtain this information would be to use the computer aided design (CAD) database, which is more standardised than CAE. However, this data depends on the company's workflow to maintain the link between the CAE and CAD models, which has yet to be well established. In addition, these databases lack information on the dependencies of the part connections, i.e. all parts are connected without any hierarchy. This hierarchy is essential for defining the direction of the edges and for identifying the vertices of the graph as either dead ends or capable of allowing energy to flow through the structure. As a result, we are looking for a method to perform this intelligently using the FE model, based on the location and closeness of parts therein.

The FE model contains mesh faces and volumes with different entities representing the connections. The mesh is defined by nodes and elements, where the element size defines the resolution of the discretization. The nodes can represent the vertices and the elements define the edges for a graph defined as $G(V, E)$ with vertices and edges. Consequently, a FE model mesh itself represents a graph. However, this graph has drawbacks. A small element size, three to five mm, for a complete vehicle will result in a large number of vertices, up to 20 million, which is computationally expensive for graph machine learning (GML) and the lack of semantics makes it difficult to analyse engineering concepts. Coarsening the crash FE mesh is an alternative, which is a topic in FE modelling [15–17]. However, rather than focusing on post-processing aspects, these studies have mainly focused on reducing the compute time of the FE simulation. Nevertheless, the result will still be a disconnected graph because a FE model contains multiple meshes whose connectivity is not element based. Therefore, we focus on linking FE entities to extract the structure of a vehicle as a connected graph.

To determine the connectivity, we split the graph extraction problem into two steps. First, component-level connectivity and then connectivity of parts within a component. Thereby we keep hierarchy information in the graph structure. Previously, we introduced a grouping method for identifying components [18]. Here, we extend this method to search for connections between components. In addition, we add edges to the graph that connect parts that belong to the same



(a) Each part as a box with its centre of gravity

(b) Group parts as a component

(c) Edges connecting components

(d) CBG

(e) sPBG

(f) mPBG

Fig. 1: Abstracted visualization of the stages for graph extraction. While the method works in 3D, we here show a 2D visualisation. Solid squares: part, dashed square: component, circle: part box center of gravity (COG), triangle: component box COG, green edges: component to component, blue edges: component to part, red edges: part to part.

component. To include timing in the graph, we also investigate to add a timing segmentation based on the timing of outgoing edges, see Section 4.2.

We consider the parts of the FE entities as vertices of the structural graph of the vehicle, which follows the scheme of [5]. We want to detect the edges that resemble the structural connectivity of the vehicle, and we propose three scenarios to do this: CBG, sPBG and mPBG. We need to extract information from the structure of the vehicle to obtain the connectivity between parts. To do this, we create 3D axis-aligned boxes for each part that contain the volume of the part's geometry, Figure 1a. Then, based on the overlap of the boxes, we define rules to group them as components, Figure 1b, and later form the structure of the graph from the overlap of the boxes. In the following three subsections, we will discuss the detailed differences between these methods and for now only describe the general idea. The CBG follows two steps: finding the connections between components, Figure 1c, and then determining the part connectivity in each component, Figure 1d. sPBG and mPBG have additional steps to convert component connections to part connections, which requires identifying the parts involved in the connectivity that supports the energy flow. We explore two scenarios for this as single and multi-part-based graphs, Figures 1e and 1f, respectively. For all these methods, we consider a directed graph whose directions are set to have a positive inner product with the impact axis, direction x in Figure 1a and Algorithm 1.

Algorithm 1 edge direction for vertices A and B, impact direction x

```
Input: TVL: Threshold Limit Value
  if $\|AB\| < TLV$ then
      if $\overrightarrow{AB} \cdot \overrightarrow{x} > 0$ then
          connect $A$ to $B$
      else
          connect $B$ to $A$
      end if
  end if
```

3.1 CBG

The construction of CBG requires first the detection of the components and then the detection of the connections between components. The component detection considers each part to be a box, then groups them together as a component, and finally evaluates the component box. For CBG, in addition to the part vertices, we also introduce component vertices into the graph. The location of these vertices is at the centre of the components and the component parts are connected to them. For example, in Figure 1a with eight parts, four components are detected and corresponding component boxes are generated, in Figure 1b. Then, using a threshold value (TLV), our algorithm searches for immediately adjacent components. The thresholding allows having several neighbours. The search algorithm sorts components by impact direction, starting from the impactor/barrier position and moving into the vehicle along the impact direction, e.g. x in Figure 1c. Finally, we connect all of the parts in each of the components to the component box.

The result at this stage, Figure 1d, is a connected graph, which is a heterogeneous graph of parts and components. Evaluating the longest path for a heterogeneous graph requires additional evaluation of edge features between vertices of different types. Therefore, our goal is to modify this graph into a homogeneous graph. First, we consider only the components as vertices, delete the vertices of the parts, and evaluate the features of the component vertices based on the parts, as we introduced earlier in [18]. This graph is CBG and doesn't contain the detailed features of all the parts. Another approach is to use the heterogeneous graph as an input to find further connectivities of the parts. We explore this approach in Sections 3.2 and 3.3.

3.2 sPBG

The sPBG is a basic approach to convert the heterogeneous part-component graph into a part graph by transferring the component vertex and its corresponding edges to a part vertex. Because of the single part selection, we call it sPBG and we consider an alternative multiple part scenario in Section 3.3. There are several ways to determine the corresponding part for each component. First, we use a simple scenario and select the largest part, the geometric aspect of the component, as the corresponding vertex for the component connection. For example, in Figure 1e with this consideration, the ○- 1 -○ remains in the same position as the component-part graph because the connecting components contain a single part. The edges ○- 2,4,5 -○ move from the component box to the largest part, so ○- 6 -○ is removed. Finally, the edge ○- 7 -○ disappears in the last components and edges ○- 8,9 -○ move to the other end of edge seven.

The sPBG graph is characterised by having a main connection from the beginning to the end of vehicles with several dead ends for each master part. We expect that the identification of the energy flow of the simulation will be limited by the existence of many dead ends. Furthermore, for sPBG a single part is the representative of a component and therefore only a single part interacts with the other parts, which in some cases is not appropriate. For example, the side-member, which is a thin-walled structure, has two U-sections welded and several reinforcement plates. In this example, information about the interactions of the other U-profiles and reinforcement plates will be missed if only one part is considered to represent the component. Next, we consider multiple connections between the components with mPBG. Multiple connections reinforce the lack of internal connections compared to sPBG.

3.3 mPBG

The mPBG is an alternative to sPBG by allowing multiple representatives for components. This approach allows for part interactions in the components and between components. Here we transfer and distribute the component vertices using the information from the component discovery process, rather than selecting the largest box. As described in [18], our component detection algorithm has two scenarios for identifying the components: full and partial overlap merge. Full overlap means a box is completely within the parent box, whereas partial overlap addresses partially overlapping scenarios. These two scenarios are treated differently for mPBG extraction. In the case of a full merge, the part is connected to its parent box, similar to sPBG. However, in partial overlap scenarios, both boxes will represent the component. In this case, a component vertex is transferred to all partially overlapped boxes. Nevertheless, each part will retain its connections to the child based on full merges. Figure 1f visualises these two scenarios. The edge ○- 2,3 -○ branches to two edges ○- 2_1,2_2 -○ and ○- 3_1,3_2 -○ respectively compared to the sPBG due to a partial merge. Furthermore, the edge ○- 9 -○ branches to ○- 9_1,9_2 -○ since it is added after the partial merge and belongs to both parent boxes.

4 Load-path detection

Understanding how an external load is transferred to a given structure helps to evaluate the performance of different components, improve structural strength and reduce structural weight in structural design and optimisation. The so-called load-path of a component is a concept for tracking the transferred load within a structure, starting from the load points and ending at the support points, which has been studied in structural design for several years [19]. Reviews of different approaches to load-path detection are proposing a new metric to find detailed load-paths at mesh size for better component design. However, we are interested in the load-path in the context of crash analysis, which involves the interaction of several components. Load-paths are typically defined as vehicle parts capable of generating resisting forces during a crash event [20]. To identify load-paths during a crash, nine load-paths were first defined and classified in [20]. These can be easily examined for signs of loading after a crash. On the other hand, this work mainly introduces new measures for evaluating real crashes.

We aim to identify the load-path to be able to compare simulations by highlighting the importance of different paths during the crash. We use the longest path calculation[4] to find the load-paths involved in absorbing the crash energy. In this calculation, we aim to look at the internal energy absorption of the parts since manufacturers optimise the energy absorption capabilities of the load-paths [20]. To achieve this, we use the so-called internal energy (IE) features introduced in [14]. Initially, one has an unweighted graph with IE features for vertices. An essential step is to convert vertex features into edge weights. In this way, the edge weights hold the absorption characteristics and instead of the longest unweighted path, we compute the potential load-path.

In the following subsections, we first introduce the edge weights as a single feature of the internal energy flow, f_{IE}, and the time segmentation, s_t. f_{IE} is computed from the vertices maximum absorbed internal energy (IE_{max}) using internal energy flow calculation, see Section 4.1. For s_t we update the graph with time segmentation to have absorption time features on the edges, see Section 4.2. Finally, in Section 4.3 we will present several ways to combine edge features.

4.1 Internal energy flow

We consider the flow equation for the propagation of the internal energy maximum (IE_{max}) feature from the vertices to the edges, f_{IE}. Our graph is a directed weighted graph $G(V, E)$ with vertices V, edges E and a weight $w(e)$ assigned to each edge. We assume that the energy flow from vertex i to j, $w_{i,j}$, is represented by an edge weight between vertices i and j. The energy flow equation relates the absorbed internal energy IE_j of a vertex v_j to the balance of the input and output IE from that vertex to its neighbours:

$$IE_j = \sum_{n \in I(j)} w_{n,j} - \sum_{n \in O(j)} w_{j,n}. \tag{1}$$

For a vertex v in a graph, we denote by $I(v)$ and $O(v)$ the set of in-neighbours and out-neighbours of v, respectively. We start computing edge weights with vertices that only have incoming edges, called dead ends. We compute the flow from the dead ends, backwards along their edge directions, to find the inflow of the dead

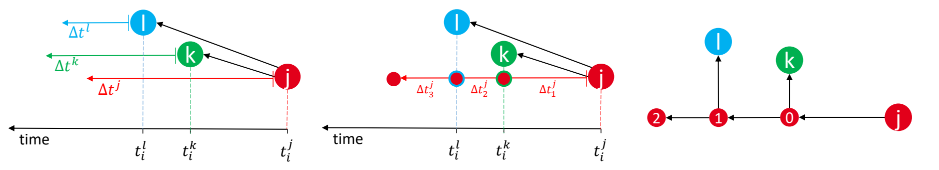
[4] The longest path in a directed acyclic graph, `dag_longest_path()`, from NetworkX

ends vertices. The active vertices for the next step calculation are the source vertices to the dead ends. Consequently, if all their outflow energy is available, we can find the inflow energy to the active vertices. Until all its outflows are known, a vertex is withheld from being an active vertex. In addition, there is a different treatment for the dead ends at vertices that have an inflow degree of zero. These source-only vertices reflect where the impact is initiated and where accordingly the kinetic energy input takes place. Therefore, these vertices are not considered when they are marked as active vertices. Instead, the edge weights of these source-only vertices are calculated when their outgoing neighbours are the active vertices. In some cases, the weights of all their outgoing edges have already been evaluated, but the active vertices may have more than one incoming edge. In this case, the energy flow is partitioned to the in-degree, $I(v)$. An unequal stiffness of the structure does not allow an equal distribution. Therefore, equal partitioning can lead to errors in the flow calculation, which we discuss in 5.1.

4.2 Time segmentation

To convert the vertex absorption times into edge weights is more complex than the handling of IE_{max}. This is because the graph connectivity of the vertices differs from the time sequence of the parts that absorb energy. Moreover, the time information of each vertex is an absorption interval (Δt), initial absorption time (t_i) to final absorption time (t_n), which may overlap with one of its neighbours. In the example shown in Figure 2, we demonstrate the time segmentation for vertex j with two successors of l and k. In this figure, the absorption period of each vertex is plotted as a vector along the time axis. The overlap of these vectors highlights the need for time segmentation, see Figure 2a. To overcome this, we segment the time interval of the absorption for each vertex. The segmentation is based on the t_i value of the successors of the vertex.

Accordingly, we add vertices to the graph for each segmented time and connect each successor vertex to the vertex added for time segmentation. In this example, a vertex is added to the graph for each successor vertex, l and k, see Figure 2b. Note that if some of the successors have the same t_i, then only one vertex will be added. In addition, to include the total absorption, an extra vertex is



(a) Initial state of the graph before time segmentation. (b) Adding vertices according to the successors. (c) The graph after adding time segmentation vertices.

Fig. 2: An example of time segmentation process for a vertex j with two outgoing edges to the successors vertices of k and l. The time axis shows the t_i value for each vertex and absorption time with an arrow in front of each vertex.

added to represent the total absorption vector as the sum of $\delta t^j = \delta t_1^j + \delta t_2^j + \delta t_3^j$, see Figure 2c. Then we sort the t_i of the successor vertices to find the connection between the new vertices. Finally, the directed edges containing the time sequences and durations are added and the old edges are deleted, see Figure 2c. Additionally, we add the initial timing t_i^k as a vertex feature for the kth segment, so that all vertices have a t_i. Finally, the edge weight s_t for time segmentation is for a directed edge from m to n defined by $s_t := t_i^m - t_i^n$.

4.3 Feature combination

We consider two approaches to combine IE_{max} and the timings of part absorption. In the first approach, we modify the vertex feature IE_{max} according to the absorption time before the flow computation from Section 4.1. To do this we look at the integration of the IE curve over time, $IE\Delta t$. The start and end of the integration are set to the minimum $t_{i_{min}}$ and maximum $t_{n_{max}}$ of the absorption times, t_i and t_n, respectively, over all parts. To simplify the calculation, we divide the area under the curve IE into three zones. For each zone the area under the curve, A, is calculated:

- ○ $(t_{i_{min}},\ t_i)$ unload period, $A_1 = 0$
- ○ $(t_i,\ t_n)$ absorption period, $A_2 = IE_{max}(t_n - t_i)/2$
- ○ $(t_n,\ t_{n_{max}})$ saturated period, $A_3 = IE_{max}(t_{n_{max}} - t_n)$

The sum of these areas is the new node feature and we compute, as in Section 4.1, the combined edge weight with the flow of $IE\Delta t$, $f_{IE\Delta t}$. In the second approach, we use the time segmentation graph. For this graph, we calculate the energy absorption efficiency, $P_e = IE/\Delta t$, where $\Delta t = s_t$, see Section 4.2, and $IE = f_{IE}$, see Section 4.1.

5 Result

We use an illustrative example presented in [18] to evaluate our method. This study contains 66 simulations; each model contains 27 parts and 11 components. The model structure is the same, therefore the graph structure remains the same for all simulations. Figure 3 shows the extracted graph for CBG, sPBG and mPBG. Here, in the graph visualisation, the vertices are positioned in the centre of its part or component box. In Figure 3a for CBG, the vertices of the graph are labelled by these components. For sPBG and mPBG each vertex refers to a part in figures 3b and 3c, where the parts corresponding to the vertex of a component are coloured grey. The mPBG has additional edges compared to sPBG that are marked in red, Figure 3c. While the CBG, sPBG and mPBG graphs are the same for 66 simulations, adding the time segmentation to the graphs can change the structure for each simulation due to different time sequences. Figure 4 shows the differences in two simulations generated by time segmentation for mPBG. In the following sections, we evaluate the computation of the IE flow and the detection of the load-path.

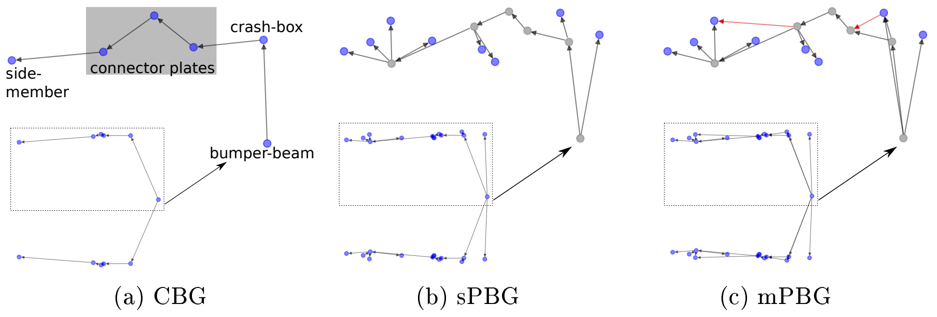


Fig. 3: Extracted graphs for the illustrative example [18]. A zoomed view of the upper half is shown for each graph[5]. The additional edges for mPBG compared to sPBG are marked as red in (c).

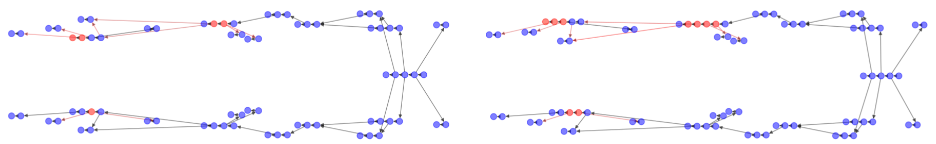


Fig. 4: mPBG segmentation differences for simulations (0) and (27) due to different times of absorption.

5.1 Graph Flow

We use the RSME of the inflow and outflow to evaluate the flow calculation as:

$$RSME = \sqrt{\frac{1}{N}\sum_{j=1}^{N} IE_j - \left(\sum_{n\in I(j)} w_{n,j} - \sum_{n\in O(j)} w_{j,n}\right)}. \tag{2}$$

The flow calculation has a small error in the order of 2 to $3e-16$ for the three graph extraction methods. The comparatively high spread of the RMSE for CBG indicates that for some simulations the connectivity of the CBG graph is limited, which increases the RMSE for these simulations.

5.2 Load-path detection

Here, we first discuss the result of the load-path detection for five reference models, as in [18], and show how the load-path detection characterises the simulations. Then, we use the best method to classify all 66 simulations. In the reference simulations – 3, 30, 31, 60, 61 — the crash-box thicknesses differ as follows. Simulation 3 has the same thickness on both left hand side (LHS) and

[5] The zoomed views use `networkx.kamada_kawai_layout()` with vertex distances and positions to improve the visualisation.

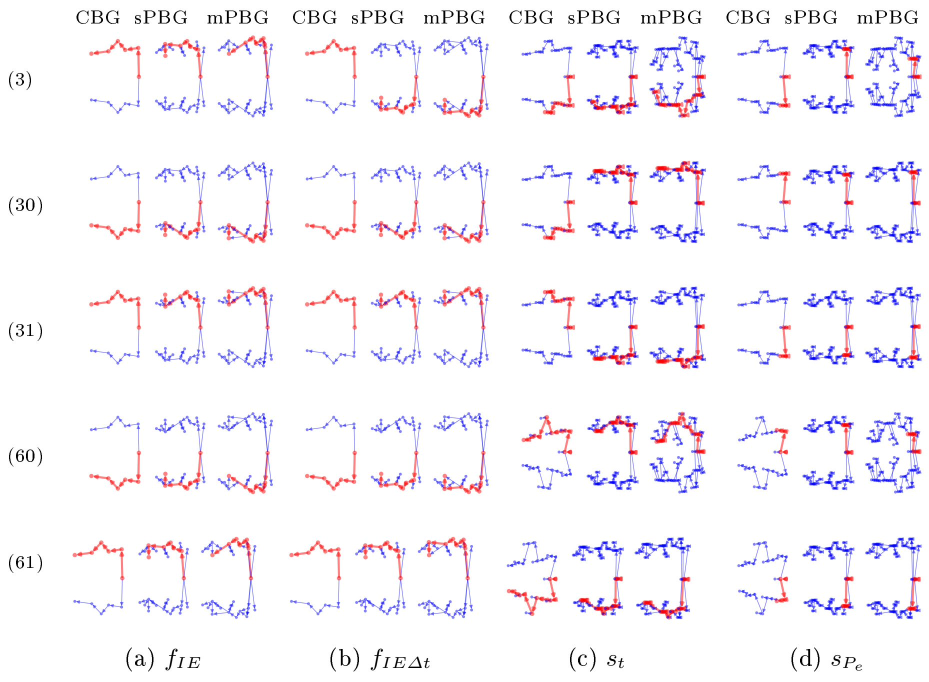


Fig. 5: Load-path detection, marked in red, for the five reference simulations from [18]. LHS and RHS are at the top and bottom, respectively. Each simulation and edge weight setup include results of CBG, sPBG and mPBG.

right hand side (RHS). Compared to 3, simulations 30 and 31 are less stiff on RHS and LHS, respectively. Whereas simulations 60 and 61 are stiffer on LHS and RHS, respectively, compared to 3.

Figure 5 summarises the load-path detection with four edge weights as described in Section 4. Columns a and c are the single feature results for f_{IE} and s_t. The other two columns are weighted with combined features $f_{IE\Delta t}$ and s_{P_e}, columns b and d respectively. We show the results of three different graph extraction methods for each scenario and the detected paths are marked in red. Based on the structural stiffness, the expected energy load-path for simulations 30 and 60 is at the RHS (bottom) and for simulations 31 and 61 at the LHS (top).

We expect that for graphs with f_{IE}-edge weight, it is the reverse of graphs with s_t-weight whether we get a top or bottom load-path. This is due to the physics of the problem, i.e. stiffer parts take more time for absorption and deform less, which means lower IE. The only exception we observe is in the result with CBG and s_t weighting. Here the detected path for these simulations does not continue to the side-member and a different side of the structure is detected

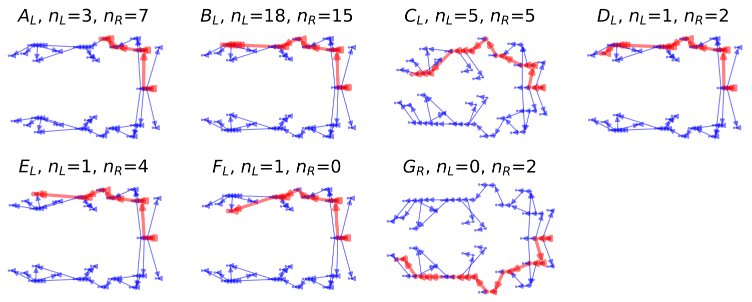


Fig. 6: Identical load-paths, marked in red, that are identified for the simulation dataset from [18]. Only one is shown if there is a symmetric pair. n_L and n_R are the number of occurrences of a load-path in the dataset, respectively.

compared to sPBG or mPBG. This example shows the limitation of CBG in time feature extraction, i.e., the component level is less sensitive than the part level.

Next for the combined features, $f_{IE\Delta t}$ and s_{P_e}, for most scenarios the detected load-path remains in the expected direction of the structure. The only exception is the CBG graph for simulation 3. Simulation 3 is a symmetric model and lacks a dominant load-path due to its symmetry. Again, the CBG method lacks the detail to realise the effect of time in detecting the load-path. The additional obvious observation is that with s_{P_e} weight the detected path is shorter. This detection describes well that the crash-box influence is much greater than those of the remaining parts. Therefore, this path captures the efficient path of the load rather than the full path along the structure.

Among these approaches, the mpBG with s_t detects the most detailed load-paths, which is better for simulation comparison. As a result, we use it to visually categorise all 66 simulations. This method categorises the data into 12 identical load-paths, where 10 are symmetric pairs, i.e., an LHS path corresponds to an RHS path. Figure 6 summarises the clusters. Most of the simulations, 33, are grouped in cluster B. The biggest difference of the clusters is between cluster A and the rest where the path ends with a crash-box absorption. The remaining clusters have similar absorption for the crash-box and differ in vertex selection for the side-member at the end of the path.

6 Conclusion and Outlook

We considered load-path detection in crash analysis, one of the automotive CAE domains, by using graph approaches. Due to the lack of graphs in the CAE data, we introduced graph extraction methods to convert the CAE analysis of crashes into graphs. To characterise the absorption path of the vehicle structure, we not only abstract the vehicle structure into the graph, but also define edge directions and edge weights. By computing the longest weighted path in a graph an automated detection of load-paths now becomes feasible. Vehicles with the same structural design have an almost similar graph structure, while edge weighting and time segmentation detect differences in load-paths. Our method showed promising results analysing an illustrative example with 66 simulations. Based on our study, it is best to use different graph extraction approaches and edge weights (w) for different applications, as follows:

a CBG, $w = f_{IE}$: crash mode analysis [18], advantage: simple and stable.
b mPBG, $w = f_{IE\Delta t}$: IE flow path analysis, advantage: more details.
c mPBG, $w = s_t$: simulation clustering using load-path, advantage: sensitivity to time sequence.
d mPBG $w = s_{P_e}$: analyse part or component efficiency.

As well as being useful for the CAE engineers, the load-path clusters from c) can also be used as labels, which opens up new possibilities for using supervised machine learning (ML) for CAE. We see as a next stage an implementation of graph embedding methods to automatically classify the results.

In addition, posture detection methods can be used to further process the data during the crash [21]. With these methods, part features should remain at the vertex level for active part detection. However, as far as we are aware, there is limited research on directed graphs to find the load-path. Furthermore, converting a whole vehicle into a graph requires additional considerations. For a complete vehicle, graph extraction can often lead to several unconnected graphs due to the existence of larger parts. Our graph extraction works for sub-models, but further heuristics are needed to extend its application, which is beyond the scope of this work. Finally, we extracted a static graph from the undeformed geometry. As the deformed structure may lead to additional contacts between parts that do not exist in the undeformed structure, it may be useful in the future to consider the deformed structures as well.

References

1. L. Qiao, L. Zhang, S. Chen, and D. Shen, "Data-driven graph construction and graph learning: A review," *Neurocomputing*, vol. 312, pp. 336–351, 2018.
2. B. Abu-Salih, "Domain-specific knowledge graphs: A survey," *Journal of Network and Computer Applications*, vol. 185, p. 103076, 2021.
3. L. Stanković, D. Mandic, M. Daković, M. Brajović, B. Scalzo, S. Li, and A. G. Constantinides, "Data analytics on graphs part III: Machine learning on graphs, from graph topology to applications," *Foundations and Trends in Machine Learning*, vol. 13, no. 4, pp. 332–530, 2020.
4. P. Schwanitz, "Towards AI based recommendations for design improvement (AI-B-REDI)." presentation at SIMVEC, Baden-Baden November 2022, 2022.
5. A. Pakiman and J. Garcke, "Graph modeling in computer assisted automotive development," in *2022 IEEE International Conference on Knowledge Graph (ICKG)*, pp. 203–210, 2022. arXiv:2209.14910.
6. J. J. Granda, "Automating the process for modeling and simulation of mechatronics systems," in *Bond Graph Modelling of Engineering Systems*, pp. 385–430, Springer, 2011.

7. J. J. Granda and T. Gloekler, “Bond graph models for reconstruction of vehicle barrier equivalent speeds,” in *Proceedings of the International Conference on Bond Graph Modeling and Simulation*, pp. 35–47, 2016.
8. P. J. Gawthrop and G. P. Bevan, “Bond-graph modeling,” *IEEE Control Systems Magazine*, vol. 27, no. 2, pp. 24–45, 2007.
9. S. G. Mentzer, R. A. Radwan, and W. T. Hollowell, “The sisame methodology for extraction of optimal lumped parameter structural crash models,” tech. rep., SAE Technical Paper, 1992.
10. J. M. Lim, “Lumped mass-spring model construction for crash analysis using full frontal impact test data,” *International Journal of Automotive Technology*, vol. 18, pp. 463–472, 2017.
11. V. A. Lange, J. Fender, L. Song, and F. Duddeck, “Early phase modeling of frontal impacts for crashworthiness: from lumped mass–spring models to deformation space models,” *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of automobile engineering*, vol. 233, no. 12, pp. 3000–3015, 2019.
12. C. Ortmann and A. Schumacher, “Graph and heuristic based topology optimization of crash loaded structures,” *Structural and Multidisciplinary Optimization*, vol. 47, no. 6, pp. 839–854, 2013.
13. D. Schneider and A. Schumacher, “Finding optimized layouts for ribs on surfaces using the graph and heuristic based topology optimization,” in *World Congress of Structural and Multidisciplinary Optimisation*, pp. 1615–1628, Springer, 2017.
14. A. Pakiman, J. Garcke, and A. Schumacher, “Knowledge discovery assistants for crash simulations with graph algorithms and energy absorption features,” *Applied Intelligence*, 2023.
15. R. E. Bank and J. Xu, “An algorithm for coarsening unstructured meshes,” *Numerische Mathematik*, vol. 73, no. 1, pp. 1–36, 1996.
16. A. Chawla, S. Mukherjee, and A. Sharma, “Mesh generation for folded airbags,” *Computer-Aided Design and Applications*, vol. 1, no. 1-4, pp. 269–276, 2004.
17. F. Montevecchi, G. Venturini, N. Grossi, A. Scippa, and G. Campatelli, “Finite element mesh coarsening for effective distortion prediction in wire arc additive manufacturing,” *Additive Manufacturing*, vol. 18, pp. 145–155, 2017.
18. A. Pakiman, J. Garcke, and A. Schumacher, “Simrank-based prediction of crash simulation similarities.” INS Preprint No. 2210, Institut für Numerische Simulation, Universität Bonn, 2022.
19. K. Marhadi and S. Venkataraman, “Comparison of quantitative and qualitative information provided by different structural load path definitions,” *International Journal for Simulation and Multidisciplinary Design Optimization*, vol. 3, no. 3, pp. 384–400, 2009.
20. M. Lindquist, A. Hall, and U. Björnstig, “Real world car crash investigations–a new approach,” *International Journal of Crashworthiness*, vol. 8, no. 4, pp. 375–384, 2003.
21. N. Ma, Z. Wu, Y.-m. Cheung, Y. Guo, Y. Gao, J. Li, and B. Jiang, “A survey of human action recognition and posture prediction,” *Tsinghua Science and Technology*, vol. 27, no. 6, pp. 973–1001, 2022.