AN ASSUMPTION NETWORK-BASED APPROACH TO SUPPORT MARGIN ALLOCATION AND MANAGEMENT

S. El Fassi, M. D. Guenov and A. Riaz
Cranfield University, United Kingdom
s.el-fassi@cranfield.ac.uk

Abstract
Presented is an approach to support margin allocation and management via a graph-theoretical network of assumptions. In contrast to the document-centric approach, the network captures assumptions dependencies, and enables an algorithmic process supporting margin allocation and management. Ultimately, this methodology is intended to assist decision-makers in managing assumptions and examining their impact on an architecture. Explicitly linking margins to assumptions allows to support mitigating their risk of invalidity. The approach is demonstrated with a conceptual aircraft design example.

Keywords: conceptual design, decision making, uncertainty, assumption management, margin

1. Introduction
The conceptual design of novel products is characterised by significant uncertainty due to lack of knowledge. Such uncertainty is referred to as epistemic uncertainty (Kiureghian and Ditlevsen, 2009), and can impede the design process. In order to proceed with the design, assumptions are introduced to fill the knowledge gap. However, the uncertainty inherent in the assumptions constitutes a risk that has to be mitigated, especially at early-stage design where about 70% of the budget is committed (INCOSE, 2015). In this regard, design margins are traditionally assigned as a risk mitigation strategy (Eckert et al., 2019).

Although the current systems engineering processes (e.g. INCOSE-TP-2003-002-04 (INCOSE, 2015), ISO/IEC/IEEE 15288 (ISO, 2015), and NASA SP-2016-6105 (NASA, 2016)) already support working with assumptions by explicitly documenting and reviewing them as the design progresses, this traditional document-centric approach is not suited for innovative design which is likely to require making an intractable amount of assumptions. In fact, a recent study that surveyed practitioners showed that simply documenting explicit assumptions has little benefit (Sadlauer et al., 2017). Furthermore, lack of formal and systematic allocation and management of design margins in practice can lead to assigning margins that are either too optimistic, such that there is an increased risk of major re-design, or too conservative, such that the system becomes over-designed, thus adversely impacting its performance.

Thus, a novel approach is proposed in this paper to support design margins allocation and management via a graph-theoretical network of architectural assumptions. The scope is restricted to the conceptual design stage, and systematic assumption maintenance following the acquisition of new knowledge, or when new assumptions are made, forms part of future work.
In the rest of the paper, background information and the state-of-the-art are presented in Section 2. The proposed approach is described in Section 3. Section 4 demonstrates the approach with an aircraft conceptual design use case. Finally, conclusions are drawn in Section 5 along with stating future work.

2. Background and state-of-the-art

2.1. System architecting and the RFLP paradigm

Complex systems design is characterised by defining a system architecture, which is “the embodiment of concept, the allocation of physical/informational function to the elements of form, and the definition of relationships among the elements and with the surrounding context” (Crawley et al., 2016). Functional reasoning, i.e. defining the functions to be performed by the system, plays a central role in system architecting (Umeda and Tomiyama, 1997). One approach to functional reasoning is the RFLP paradigm (Kleiner and Kramer, 2013), which is based on the VDI 2206 standard Design methodology for mechatronic systems (VDI, 2004). It considers that functional reasoning is distributed over four notional domains: Requirements, Functional, Logical and Physical, which can be defined as follows:

- Requirements domain (R): contains the requirements which can be hierarchically decomposed. The requirements are mapped to the functions that fulfil them in the Functional domain.
- Functional domain (F): contains the functions that the system must perform, which can also be hierarchically decomposed. The functions are mapped to the components that realise them.
- Logical domain (L): contains the solutions (components) realising the system functions, in addition to their connectivity (via ports).
- Physical domain (P): contains a 3D CAD design model which essentially captures the spatial/topological relationships amongst the components.

Recently, Bile et al. (2018) proposed to augment RFLP with a Computational domain (C) for automated systems sizing, followed by a graph-theoretical structure that captures the dependencies between the RFLC domains (Guenov et al., 2020).

Architectural decisions are in fact the most impactful design decisions, given that most of the budget is committed at the early stage. Therefore, it is of the utmost importance to examine their sensitivity to the assumptions (Crawley et al., 2016). The following section reviews architectural assumptions and the existing documentation and management approaches.

2.2. Architectural assumptions

It is acknowledged that many definitions of the concept assumption exist, in addition to the fact that the distinction between closely related concepts such as axiom, premise, or presupposition varies throughout the literature (Berner, 2017). The most commonly adopted definition of assumption remains the dictionary one, e.g. “a thing that is accepted as true or as certain to happen, without proof” (Oxford English Dictionary), or “something that you accept as true without question or proof” (Cambridge Dictionary). A similar definition is adopted in the field of artificial intelligence, where an assumption is “something which is accepted in the absence of evidence to the contrary” (Ramsey, 1988).

However, such definitions are incomplete as they do not capture some essential characteristics of assumptions. Some of these characteristics were defined by Yang et al. (2018) in the context of software development, where assumptions are: (i) subjective, i.e. can be seen as assumptions by some stakeholders, or design decisions by others, (ii) related to other software artefacts, such as requirements or components, (iii) dynamic, i.e. evolve with time, and (iv) context-dependent, i.e. could be valid in one project, and invalid in another. Furthermore, assumptions are inherently uncertain, and the degree of confidence in making them varies based on the strength of background knowledge.

Therefore, the concept of assumption may not have a concise definition as it has many characteristics, but rather requires a definition that reflects its many aspects. Thus, the following definition is proposed:

An assumption is a context-dependent belief, with a varying degree of confidence, that requires justification to become knowledge. An architectural assumption bridges the gap between available knowledge and knowledge required to proceed with the conceptual design.
Formal and semi-formal methods have been developed to manage assumptions in the field of software engineering. Lewis et al. (2004) proposed the Assumption Management System that allows to extract and record software assumptions from Java source code into a repository. However, the assumptions have to be recorded as structured comments within the source code, and assumptions capturing and management are conducted after the architecture is defined. Tirumala (2006) proposed an assumptions management framework for software systems which sets some policies on assumption selection and validation, i.e. “a relevant subset of assumptions can be validated or flagged as invalid automatically as the system evolves”. Although such a capability allows reducing the cost of managing assumptions, this framework is restricted to assumptions on software components, meaning that the dependencies between assumptions and other architectural elements (such as requirements) are not considered. Recently, Yang et al. (2018) proposed another approach which includes a documentation framework that records dependencies both amongst assumptions, and between assumptions and other software artefacts. Such knowledge about dependency is crucial in assessing the impact of assumptions on the overall architecture, but the lack of support for capturing such dependencies highly increases the effort needed for systematic assumption management. Moreover, the lack of assumption’s uncertainty assessment in this framework (and the other frameworks) could be actually misleading the decision-makers into thinking that all assumptions are made with the same degree of confidence, in addition to preventing from prioritising the significant amount of assumptions in the case of innovative design. Compared to software engineering, there is a lack of systematic approaches to manage assumptions in physical systems design. This results in relying on conservative margins to manage epistemic uncertainty. Such lack may be explained by three main factors. First, the traditional document-centric approach of physical systems engineering, as opposed to software architectures being readily available in a computerised format. Second, the traditional, less rigorous representation of physical systems architectures, as opposed to software architectures being readily represented in formal logic, thus impeding artificial reasoning. Third, the fact that design margins pertain to the physical domain, thus transcending the realm of software engineering.

2.3. Margin allocation and management

Traditionally, margins have been used as a strategy to mitigate the risk associated with epistemic uncertainty. There exist many terms and definitions for the concept of margin. One definition was proposed by Eckert et al. (2019) where a design margin is the extent to which the value of a design parameter exceeds what is needed to fulfil the requirements, thus accounting for the uncertainties in addition to providing flexibility. Effectively, the designed system becomes more capable, withstands worse environments, and lasts longer than necessary (McManus and Hastings, 2005). Although margins are typically assigned in a highly conservative manner in practice, some formal methods based on probabilistic allocation have been developed, such as Thunnissen (2004) who proposed a six-steps method based on propagating the design variables’ uncertainty through a Monte Carlo simulation, and then a margin is allocated based on the 95th, 99th or 99.9th percentile of the probability distributions of the outputs of interest. Zang et al. (2015) proposed another probabilistic strategy where margins are assigned to aircraft sizing and performance models. Basically, a set of random outputs is obtained by sampling uncertain design variables, from which probabilities of constraint satisfaction are calculated. The strategy is then to satisfy the probability of success while optimising a figure of merit. The same strategy is then used in the “Sculpting” approach (Cooke et al., 2015) to generate parallel coordinates plots that assist decision-making about margin allocation. However, assuming subjective probability distributions, especially in the presence of high epistemic uncertainty characterising conceptual design, remains an issue. In fact, the arbitrary and unjustified use of probability distributions due to lack of knowledge has been recognised as a common flaw in aerospace engineering (Zaidi et al., 2014). Thus, it prompts us to question how reliable such probabilistic strategies are in the first place. Furthermore, probabilistic approaches treat computational models independently of the related architectural domains, as opposed to the RFLP(C) paradigm presented in Section 2.1.

Other limitations have been discussed by Eckert et al. (2019) which include the lack of a method for allocating suitable margins, as well as the lack of tracking tools to capture margins and their rationale.
Margins are assigned to parameters in order to mitigate the risk associated with epistemic uncertainty. Knowing that assumptions are made to fill the knowledge gap characterising epistemic uncertainty, it is argued in this paper that margins should explicitly mitigate the risk associated with architectural assumptions being invalid, by providing flexibility in order to absorb resulting architectural changes, while the assumptions themselves would represent the rationale behind margin allocation. This leads to the proposed approach in Section 3, which would constitute an alternative to the probabilistic strategies.

3. Proposed approach

Figure 1 illustrates the flowchart of the proposed approach, where the first step is to capture and codify architectural assumptions using an object-oriented approach, then the uncertainty inherent to each assumption is assessed based on the strength of the background knowledge, followed by capturing the dependencies both amongst assumptions, and between assumptions and other architectural elements. All the aforementioned steps lead to generating a graph-theoretical assumption network which, as opposed to the traditional document-centric approach, is compatible with the Model-Based Systems Engineering methodology. The network is to be maintained when new knowledge is acquired in order to keep consistent and up-to-date records of the assumptions made, which would be accessible to all stakeholders. Finally, the network is used as an input to support margin allocation and management.

As stated earlier, systematic network maintenance is out of the paper’s scope as it forms part of future work. For the time being, maintenance consists of capturing new assumptions, and manually updating the status, confidence and dependencies of existing assumptions in order to enable Algorithm 2 (step 4).

3.1. Step 1: capturing architectural assumptions

To capture and codify assumptions, an object-oriented approach is adopted, as illustrated by the data structure in Figure 2. The \texttt{<<assumption>>} attributes are described as follows:

- \textit{status}: a string that can take one of the following values \{Awaiting evaluation, Valid, Invalid\}.
- \textit{description}: a textual description of the assumption, as well as the rationale behind it.
- \textit{confidence}: a string that represents the level of confidence in making the assumption, and can take one of the following values \{High, Moderate, Low\} in Step 2.
- \textit{dependency}: an array that stores the relationships captured in Step 3.
3.2. Step 2: assessing assumptions’ inherent uncertainty

Although assumptions are made to fill the knowledge gap, i.e. dealing with epistemic uncertainty, they themselves are inherently uncertain due to the varying strength of the background knowledge. Therefore, reasoning with assumptions in order to assess risks and allocate margins as a mitigation strategy has to be supported by assessing the associated uncertainty.

For this purpose, the attribute confidence is used to reflect the degree of confidence in making the assumption, based on the strength of the following background knowledge aspects: Data, Models, and Expert agreement. For this purpose, the guidelines proposed by Flage and Aven (2009) are adopted:

- confidence = High if all of the following conditions are met: (i) Much reliable data are available; (ii) The phenomena involved are well understood, the models used are known to give predictions with the required accuracy; and (iii) There is broad agreement among experts.
- confidence = Low if one or more of the following conditions are met: (i) Data are not available or are unreliable; (ii) The phenomena involved are not well understood, high fidelity models are not applicable / models are non-existent or believed to give poor predictions; and/or (iii) There is a lack of agreement among experts.
- confidence = Moderate as an intermediate state, e.g.: (i) Some reliable data are available; and (ii) The phenomena involved are well understood, but the models used are considered simple.

The value of the attribute confidence is to be assigned manually based on the architect’s assessment.

3.3. Step 3: capturing assumptions’ dependency

We distinguish between two types of dependency: Inter-domain dependency, which refers to the relationships between assumptions and other architectural elements (e.g., requirements), and Intra-domain dependency, which refers to the relationships amongst assumptions.

3.3.1. Inter-domain dependency

The graph-theoretical structure proposed by Guenov et al. (2020) is used as part of the proposed approach in order to evaluate the impact assumptions have on the different architectural domains (RFLC), where R contains the Requirement objects $\rho_i$, F contains the Function objects $\phi_i$, L contains the Component objects $\sigma_i$, and C contains both the Model objects $\mu_i$ and the Parameter objects $p_i$.

To capture the inter-domain dependencies, the aforementioned graph-theoretical structure is extended by introducing an Assumption domain (A) containing the Assumption objects $\alpha_i$.

The following inter-domain dependency links are then defined:

- $\alpha_i \leftrightarrow \rho_i$: occurs when interpreting top-level requirements or deriving new requirements.
- $\alpha_i \leftrightarrow \phi_i$: occurs as part of the rationale behind defining the functions.
- $\alpha_i \leftrightarrow \sigma_i$: occurs as part of the rationale behind selecting the solutions.
- $\alpha_i \leftrightarrow \mu_i$: refers to the assumptions associated with the computational model selection.
- $\alpha_i \leftrightarrow p_i$: occurs when assigning a value to an uncertain parameter.

The proposed links form the edges $E_i$ of 5 undirected bipartite graphs: $G_R(\alpha, \rho, E_R)$, $G_F(\alpha, \phi, E_F)$, $G_C(\alpha, \sigma, E_C)$, $G_M(\alpha, \mu, E_M)$, $G_P(\alpha, p, E_P)$ corresponding to $\alpha_i \leftrightarrow \rho_i$, $\alpha_i \leftrightarrow \phi_i$, $\alpha_i \leftrightarrow \sigma_i$, $\alpha_i \leftrightarrow \mu_i$, and $\alpha_i \leftrightarrow p_i$, respectively.
The relationships between the architectural elements Requirement, Function, Component, Model and Parameter are already captured by the graph-theoretical structure proposed by Guenov et al. (2020).

### 3.3.2. Intra-domain dependency

When an assumption is dependent on another one, it can either strengthen or weaken it as it is validated. Therefore, we distinguish between two types of intra-domain dependency links:

- **Assumption i (Aᵢ) constrains Assumption j (Aⱼ):** if confidence in Aᵢ increases, confidence in Aⱼ increases as a result. This is a strengthening relationship that can be attributed to two sources: the first source is architecture definition under the RFLP(C) paradigm, e.g. selecting a solution dictates which computational model to use, thus an assumption on the solution would automatically constrain an assumption on the corresponding model. The second source is assumption derivation, i.e. Aᵢ causes Aⱼ. This constitutes a directed edge (from Aᵢ to Aⱼ) in the assumption network.

- **Aᵢ conflicts with Aⱼ:** at least one of the two assumptions is invalid. This is a weakening relationship since an increased confidence in one assumption increases the likelihood of the other assumption being invalid. This constitutes an undirected edge in the assumption network.

The proposed links form the edges of a mixed graph MG(α, Eᵣ, Eⱼ), where Eᵣ refers to the edges of the constraint link, and Eⱼ refers to the edges of the conflict link. This provides an approach to capture dependencies amongst assumptions, and simplifies the Relationship viewpoint (Yang et al., 2017) by proposing two, mutually exclusive relationships.

### 3.4. Step 4: allocating and managing design margins

This section describes the algorithmic process that supports design margins allocation and management.

#### 3.4.1. Algorithm 1: margin allocation support

Algorithm 1 enables allocating margins with respect to sets of assumptions for which they are mitigating their associated risk, in addition to recording the margins within a list (Algorithm 1, Lines 1-5). Thus, it enables to capture margins, and their rationale that consists of the associated assumptions. Then, a rule is tested to check if a margin addresses assumptions corresponding to only one of the following: (1) uncertain parameters which values are assigned by the architect (i.e. independent variables, or root nodes in the graph), subject to performance requirements; (2) outputs of uncertain models (i.e. dependent variables, or intermediate and leaf nodes in the graph) (Algorithm 1, Lines 6-14). The aforementioned rule is aimed at reducing margin redundancy.

---

Algorithm 1: Margin allocation support

**Input:** Assumptions {αᵢ}, Margins {Mᵢ}, Eᵣ and Eⱼ (from Step 3)

**Output:** Set of allocated margins: $M = \{M_1, M_2, \ldots, Mₙ\}$, set of assumptions $αⱼ$ addressed by margin $M_i$: $A_M = \{α₁, α₂, \ldots, αₚ\}$

1. Create two empty lists M and $A_{M_i}$
2. Add allocated margins to M
3. ∀ $M_i \in M$, add assumptions $αⱼ$ mitigated by margin $M_i$ to $A_{M_i}$
4. If $∃ αⱼ: αⱼ \notin A_{M_i}$ then
   - Notify user to mitigate the risk associated with $αⱼ$
5. Let μ be a model, where x is an input and y is an output. Let $M_i$ be the margin assigned to y
6. If hasParentNode(x) = True then
   - x is a dependent parameter
   - If $∃ αⱼ: αⱼ \notin A_{M_i}$, $αⱼ \rightarrow \mu \notin Eⱼ$ then
     - Notify user that $M_i$ cannot mitigate the risk associated with $αⱼ$
8. Else
   - x is an independent parameter
9. If $∃ αⱼ: αⱼ \notin A_{M_i}$, $αⱼ \rightarrow x \notin Eᵣ$ then
   - Notify user that $M_i$ cannot mitigate the risk associated with $αⱼ$
10. Return $M$, and $\{A_{M_i} \forall M_i \in M\}$
3.4.2. Algorithm 2: margin management support

Algorithm 2 enables updating previously assigned margins when the assumption network is updated following a maintenance (e.g. some assumptions are validated, or conflicts are resolved). The algorithm consists of four rules which are checked to notify the user about the affected margins to be reviewed. 

**Rule 1** consists of monitoring the confidence and status of assumptions so that, as their uncertainty decreases, their associated margin must be reduced. **Rule 2** states that if the confidence in a constraining assumption increases (or is validated), then the margin associated with the constrained assumption must be reduced. **Rule 3** states that if an assumption $a_1$ conflicts with $a_2$, and the confidence in $a_1$ increases, it implies that the likelihood of $a_2$ being invalid increases. Consequently, the margin associated with $a_2$ must be increased. Moreover, if a conflict between assumptions is resolved, the margins associated with these assumptions must be reduced as a result of the lower risk of invalidity. **Rule 4** states that if a new assumption is added to the set associated with an existing margin, then that margin must be increased.

Such an approach also enables to capture the rationale associated with changes to the margins.

Algorithm 2: Margin management support

<table>
<thead>
<tr>
<th>Input: $M$, ${A_M, \forall M_i \in M}$, $E_{cons}$ and $E_{conf}$ (from Step 3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: Notification to update $M$</td>
</tr>
<tr>
<td><strong>//Changes in the attributes of $\forall a_i \in A_M, \forall M_i \in M$ trigger the following rules</strong></td>
</tr>
<tr>
<td><strong>//Rule 1</strong></td>
</tr>
<tr>
<td>1. if (confidence($a_i$) increases)</td>
</tr>
<tr>
<td>2. Notify user to reduce $M_i$</td>
</tr>
<tr>
<td>3. else if confidence($a_i$) decreases then</td>
</tr>
<tr>
<td>4. Notify user to increase $M_i$</td>
</tr>
<tr>
<td>5. else if status($a_i$) = “Invalid” then</td>
</tr>
<tr>
<td>6. Notify user to revisit $M_i$, depending on new assumption</td>
</tr>
<tr>
<td><strong>//Rule 2</strong></td>
</tr>
<tr>
<td>7. if ($a_i = E_{cons}(x,2)$, $\forall x$) &amp; [(confidence($E_{cons}(x,1)$) increases)</td>
</tr>
<tr>
<td>8. Notify user to reduce $M_i$</td>
</tr>
<tr>
<td>9. else if ($a_i = E_{cons}(x,2)$, $\forall x$) &amp; (confidence($E_{cons}(x,1)$) decreases) then</td>
</tr>
<tr>
<td>10. Notify user to increase $M_i$</td>
</tr>
<tr>
<td><strong>//Rule 3</strong></td>
</tr>
<tr>
<td>11. if ($a_i = E_{conf}(x,1)$, $\forall x$) &amp; (confidence($E_{conf}(x,2)$) increases) then</td>
</tr>
<tr>
<td>12. Notify user to increase $M_i$</td>
</tr>
<tr>
<td>13. else if $a_i$ no longer belongs to $E_{conf}$ then</td>
</tr>
<tr>
<td>14. Notify user to reduce $M_i$</td>
</tr>
<tr>
<td><strong>//Rule 4</strong></td>
</tr>
<tr>
<td>15. if a new assumption is added to $A_M$ then</td>
</tr>
<tr>
<td>16. Notify user to increase $M_i$</td>
</tr>
</tbody>
</table>

4. Demonstration

The proposed approach is demonstrated with the following use case. Let us consider the conceptual design of a lightweight fighter aircraft to replace the Lockheed Martin F-16, as described by Raymer (2018). The new design is characterised by the use of a novel (unproven) capability that transforms the shape of the tail from a “V” shape to a vertical tail, depending on the flight speed, for stability and control. Additionally, the design includes a novel engine, with a low Technology Readiness Level (TRL), to reduce fuel consumption. Assumptions made during the conceptual design stage are listed in Table 1, and captured using the proposed data structure as illustrated in Figure 3. The status of all the assumptions is initially set as the default value ‘Awaiting evaluation’, and the confidence values are arbitrarily assigned for the sake of demonstration. After capturing the individual assumptions and assessing their uncertainty (steps 1 and 2, respectively), the assumptions’ dependencies are then captured as edges of a graph-theoretical network, as illustrated in Figure 3. Both types of dependency, i.e. inter-domain and intra-domain, are demonstrated. For instance, the structure is assumed to be made of composite material, which translates to the inter-domain edge $a_8 \leftrightarrow \sigma_4$ (refer to Figure 3). Deciding to go with a low TRL engine led to assuming an existing engine for the purpose of sizing and analysis, which in turn led to assuming a particular value of the Specific Fuel Consumption to adjust for the advanced
technology. These assumptions translate to the constraint edges α₃ → α₄ and α₄ → α₅, respectively. Additionally, assumption α₁₀ states that 83% of the fuselage volume is to be occupied by bladder fuel tanks, whereas α₁₁ states that 20% of the fuselage volume is to be occupied by the missiles bay. 83% + 20% > 100% implies there is a conflict, which translates to the conflict edge α₁₀ ↔ α₁₁.

<table>
<thead>
<tr>
<th>Assumption</th>
<th>Description</th>
<th>Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>α₁</td>
<td>Tail convert function</td>
<td>Convert from V-tail (subsonic) to vertical tail (supersonic)</td>
</tr>
<tr>
<td>α₂</td>
<td>Unproven tech</td>
<td>Unproven technology assumed to be feasible</td>
</tr>
<tr>
<td>α₃</td>
<td>Rubber engine</td>
<td>Low Technology Readiness Level, rubber engine is assumed</td>
</tr>
<tr>
<td>α₄</td>
<td>Engine for analysis</td>
<td>Existing turbofan engine is assumed for sizing and analysis</td>
</tr>
<tr>
<td>α₅</td>
<td>SFC adjustment</td>
<td>Assumed reduction of 20% in the Specific Fuel Consumption (SFC) to adjust for advanced technology</td>
</tr>
<tr>
<td>α₆</td>
<td>Airfoil</td>
<td>Airfoil of type 64A006 is assumed</td>
</tr>
<tr>
<td>α₇</td>
<td>Stat model suitable</td>
<td>Statistical model assumed to be suitable</td>
</tr>
<tr>
<td>α₈</td>
<td>Composite structure</td>
<td>Composite structure is assumed as a solution</td>
</tr>
<tr>
<td>α₉</td>
<td>Wₑ adjusted for tail</td>
<td>Empty weight adjusted by +200 lb for impact of tail</td>
</tr>
<tr>
<td>α₁₀</td>
<td>Vₑ/Vₑ</td>
<td>Assume that integral wing tanks occupy 85% of wing volume, and bladder fuselage tanks occupy 83% of fuselage volume</td>
</tr>
<tr>
<td>α₁₁</td>
<td>Vₐₑ/Vₑ</td>
<td>Assume missiles bay occupies 20% of fuselage volume</td>
</tr>
</tbody>
</table>

After creating the assumption network, the algorithmic process described in Section 3.4 is employed to support margin allocation and management. The computational model for aircraft weight estimation (μ₄) is considered for demonstration and illustrated in Figure 4, where Mₓ refers to the margin assigned to parameter x (e.g. M_AR is the margin assigned to the Aspect Ratio (AR)). Following the process defined by Algorithm 1, the allocated margins are captured and recorded as a set M={M_SFC, M_AR, M_Wₑ/W₀}, along with their associated assumptions representing their rationale, i.e. Aₘ₄=[α₄, α₅]; M_AR=[α₇] and Aₘ₄=Wₑ/W₀=[α₈, α₉]. For instance, a margin of -8% is assigned to AR, which also mitigates the risk of transonic pitch-up, and brings the initial estimate down to AR = 3.5. Moreover, as shown in Figure 4, each margin is addressing only assumptions linked through the network to the parameter or model’s output it is assigned to. Thus, there is no need to assign a margin to W₀ in order to account for uncertainty in Wₑ/W₀ and Wₑ/W₀. Otherwise, a margin on Wₑ(Mₑ), such that Aₘₑ=[α₄, α₅, α₇, α₈, α₉], would be identified by Algorithm 1 (Lines 11-14) as redundant since SFC (p₄.1) is a root node, and the statement ((α₄ ∈ Aₘₑ) & (α₄ ↔ p₄.1 ∈ Eₚ)) would become true.
Furthermore, when new assumptions are added and existing assumptions are revised as new knowledge is acquired, the rules defined in Algorithm 2 would allow to notify the lead systems engineer to revisit the affected margins accordingly, as well as to record the rationale associated with changes to margins. For instance, it is decided with certainty, later in the design process, that the structure is to be made of composite material. Thus, $\alpha_8$ is validated and transformed into a design decision. Therefore, Rule 1 of Algorithm 2 allows to notify the systems engineer to reduce $M_{W_e/W_o}$ as a result of a reduced set of assumptions to be managed: From Algorithm 2 (Lines 1-2), status($\alpha_8$) = “Valid” (i.e. $A_{M_{W_e/W_o}}$ reduces to $\{\alpha_9\}$) $\Rightarrow$ $M_{W_e/W_o}$ is to be reduced. Furthermore, the airfoil type is fixed, meaning that status($\alpha_6$) = “Valid”. Therefore, Rule 2 (Lines 7-8) allows to notify the systems engineer to reduce $M_{AR}$ as a result of validating the assumption $\alpha_6$, which was constraining $\alpha_7$.

5. Conclusions and future work

Presented in this paper is an approach to support design margins allocation and management via a graph-theoretical network of assumptions. The approach builds upon software engineering concepts and extends assumption management for software systems to physical systems through explicitly linking assumptions to design margins. The network enables capturing assumptions dependencies through the RFLP paradigm, thus allowing the examination of the impact assumptions have on the architecture. Two algorithms have been proposed to support margin allocation and management: the first algorithm provides a means to capture margins and identify instances of margin redundancy, and the second algorithm makes use of the confidence and dependency of assumptions in order to update assigned margins as new knowledge is acquired. Furthermore, the proposed approach enhances collaborative design due to its compatibility with the Model-Based Systems Engineering methodology.

Current limitations of the proposed approach include its scalability, which is affected by manual network maintenance. For instance, retracting invalid assumptions without revising all the associated assumptions and design decisions (which can be manually intractable in a large-scale project) can lead to contradictions. Thus, a systematic approach to maintain the network’s consistency is needed. Future work includes visualising uncertainty within the assumption network, developing a systematic approach to maintain the network’s consistency, and evaluating the applicability and usefulness of the proposed approach by practitioners from industry.

References

Berner, C.L. (2017), Contributions to Improved Risk Assessments: To Better Reflect the Strength of Background Knowledge, [PhD Thesis], University of Stavanger.


