Proceedings of the ASME 2020 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference
IDETC/CIE 2020
August 16-19, 2020, St. Louis, USA

IDETC2020-22559

DESIGN FOR FLEXIBILITY: A GRAPH COLORING TECHNIQUE TO STUDY DESIGN CHANGES IN THE TETHERED ECONOMY WORLD

Spenser Estrada
The Department of Mathematics and Statistics
San Jose State University
San Jose, California 95192
spenser.estrada@sjsu.edu

Emilyn Green
The Department of Mathematics and Statistics
San Jose State University
San Jose, California 95192
emilyn.green@sjsu.edu

Sogol Jahanbekam
The Department of Mathematics and Statistics
San Jose State University
San Jose, California, 95192
sogol.jahanbekam@sjsu.edu

Sara Behdad*
Environmental Engineering Sciences
University of Florida
Gainesville, Florida 32606
sara.behdad@essie.ufl.edu

ABSTRACT

Digitization, connected networks, embedded software, and smart devices have resulted in a major paradigm shift in business models. Transformative service-based business models are dominating the market, where advancement in technology has paved the way for offering not only a set of new services but also altering product functionalities and services over time. This paradigm shift calls for new design approaches. Designers should be able to design flexible products and services that can adapt to a wide range of consumer needs over time. To address the need for designing for flexibility, the objective of this study is to develop a graph coloring technique that can model changes in the functional requirements of a product and determine the minimum number of physical parts needed to meet future functionalities. This technique relies on vertex labeling by the designer and the construction of a core graph combining key elements of all desired iterations, which is then colored by label. One numerical example and one real-world example are provided to show the application of the proposed model.

*Address all correspondence to this author.

NOMENCLATURE

G A graph composed of a vertex set $V(G)$ and an edge set $E(G)$
FR Functional Requirement
$u, v, w, \ldots$ A vertex of a graph G.
n_i The number of vertices in a graph G_i
m_i The number of edges in a graph G_i
uv An edge in a graph between nodes u and v
$G-v$ The graph formed from G with node v and all of its incident edges removed
$G-X$ The graph formed from G with nodes in some set $X \subseteq V(G)$ and all of their incident edges removed
$G \setminus H$ The graph formed from G by removing a subgraph H
$G+uv$ The graph formed from G by adding edge uv
c The number of colors used in a coloring of G
$\chi(G)$ The chromatic number of G, i.e., the minimum possible value of c
k A label used to group vertices

INTRODUCTION

The term "Tethered appliances" was first used in 2008 by Zittrain [1], in his book about the future of the Internet, to refer to

an emerging class of products defined by their ability to be monitored and altered by their sellers or owners due to advancements in wireless and GPS technology. "Internet of Things" devices, smart appliances, embedded-sensor devices, and voice assistants are examples of tethered products which require a consistent connection between users and device makers. Hooftnagle et al. [2] defined "tethered products" as devices whose functionalities and future iterations rely on ongoing connections between user and producer.

Corporations have come to the understanding that a profitable approach for maintaining an ongoing connection with customers is to focus on providing a continuous service, rather than one-time purchased products. In this marketing model, manufacturers have better control over the life cycle of a product, and in fact benefit from managing issues ranging from quality and dependability of service, to end-of-use recovery, and ultimately recycling of materials. In today's IoT market, designers need specific design tools and techniques that enable them to a) model the dynamic nature of the various iterations of this new type of product and b) bundle the concepts of product/service together to create a successful commercial model that appeals to consumers.

Inspired by the discussion of Hooftnagle et al. around the concept of the "Tethered Economy" [2], the objective of this paper is to propose a graph unification and coloring technique that considers successive product iterations and gives flexibility to add or remove product functionalities over time. We include a discussion of the advantages and disadvantages of tethered products as part of the background for this technique. While the intent is not necessarily to promote the concept of tethered products, the proposed model helps designers who are working on these types of products to envision future iterations more accurately at the early design stage.

Designers do not always have a complete picture of the future iterations that will be needed during the entire life cycle of a product, and yet modelling tools that allow them to plan for many types of probable future iterations can improve the design process for tethered products. Modeling tools that provide better information about how to integrate the (sometimes competing) needs of future iterations will enable designers to make informed decisions when identifying the optimal number of parts for a product. In this study, we demonstrate how graphs - composed of vertices and edges - can be used to model the requirements of the multiple stages that a product may go through during its entire life cycle. We further develop a graph coloring and unification process to model a consensus set of functional requirements, and to help designers determine the minimal optimal number of parts or modules for the product. This approach expands on previous work done by some authors of this paper with Gopalakrishnan, et al [3] in 2019, which also developed a graph coloring technique for determining the minimum number of parts of a product. This study extends the technique to modeling products with multiple iterations, specifically in the case that functional requirements change over time.

The next section provides a review of relevant research that informs the work of this paper. Subsequent sections provide a detailed methodology, as well as examples and a discussion of possible applications.

BACKGROUND

There are several powerful disruptive trends in industry that facilitate the broad adoption of "Tethered Economy" products. First, extensive progress in the field of information technology helps corporations to trace materials anywhere in their supply chain and monitor the status of products during usage as well as the end-of-use phase [4]. Advancements in data collection and sensor-based technologies, big data analytics, distributed ledger technologies, and autonomous systems are just a few examples of novel technologies that revolutionize the way that corporations design and sell their products. Second, there is a pervasive shift in consumer behavior as younger generations of users have shown that they prefer access over ownership [5]. Third, the emergence of new business models paves the way for shifting toward service-based strategies [6], where businesses switch to subscription and membership models as opposed to selling the ownership of their physical products to consumers.

The above-mentioned trends work together to move the market towards the tethered economy. This complicates purchase decisions for users, as they need to think about the future service costs of a product, its compatibility with other devices, data security, and privacy concerns. Similarly, new complications arise in the design process, as manufacturers now need to consider future iterations of products, the merging of services, and blending hardware and software features together.

According to Hooftnagle et al [2], corporations rely on two primary mechanisms to tether a product: tethering through design and tethering through law. Tethered products often have three distinctive features: (1) they depend on software codes for their operations, (2) they are equipped with data collection technologies such as sensors to facilitate communication of product life cycle data as well as consumer behavior, and (3) they require persistent network connections to enable long distance usage and control of the device [2].

While design is the main mechanism by which tethering is implemented, force of law is another strategy used for tethering. Device makers employ carefully drafted service contracts and mechanisms like copyright and patent law to influence the market. For example, companies could regulate the repair rights of consumers with the help of current data privacy laws, restrict the unlocking of smart devices for reuse and recycling, or debar consumers from using third-party repair services by adding legal terms into their warranty contracts [7]. Currently the law is limited in its power to regulate manufacturers' tethering of products. However, as legislatures and government agencies fight to

protect consumers from the potential harm of such efforts (e.g. implanted medical devices responding to remote control, automatic alteration or deactivation of appliances), it is expected that new laws and regulations will be enacted that will require manufacturers to pay special attention to the way that they design and manipulate products with multiple iterations [8].

New challenges in design arise from this paradigm shift toward ever-connected products. The ranking of functional requirements can change based on which iteration of a given product is under consideration, and additional consideration must be given to evolving customer needs over time. Bearing this in mind, the number of physical parts that are included in the product usually remains fixed over the physical life cycle of a device, while the device maker retains the ability to alter product features and functionalities through software codes. Therefore, number of physical parts is often a decision made at the early stage of the design process, despite uncertainties regarding future iterations of a product.

Zhang et al. [9] have highlighted the need for appropriate mathematical models that simulate the entire ecosystem of a product. Such models should consider product iterations over time, as well as the evolution of its components, and even the interaction of various performance attributes of a component throughout the product's life cycle. Mathematical exploration of product life cycle decisions can be conducted with the use of normative methods such as Monte Carlo simulation [10], statistical analysis, Bayesian approach [11], and probability theories [12], to name a few. In this study, we will discuss the use of graph theory and network modeling techniques for considering product requirements over time. The idea behind this paper is to consider N graphs, each representing the requirements of a specific iteration of a product. The proposed modeling approach has unique applications in different settings, including but not limited to the following scenarios:

1. **Technology Shifts:** technological evolution of a product including both functional and technological changes where new features and functionalities will be added or removed from the product over time.
2. **Product Generations:** design for a product family where platform-based product development is considered.
3. **New Users:** design for multiple iterations where needs of several users should be satisfied
4. **Updated Software:** design in the tethered economy world where the access of the original user is controlled by new upcoming business models
5. **Product Iterations:** design that covers the needs of several phases within the product life cycle, including manufacturing, initial and subsequent users, and end-of-life product recovery, such as recycling.

There are several ongoing efforts to develop techniques that can categorize and model different product requirements. Behera et al. [13] discuss the use of lattice theory for sharing design definitions across different product life cycles. According to their argument, it is feasible to visualize the bill of materials of a product as a lattice and then insert a given lattice into a complete lattice generated from the same product. They showed that by utilizing lattices, multiple Bills of Material (BoMs) can be embedded into one complete lattice which contains every possible combination of individual parts. They acknowledged that while lattice structures are helpful in providing a single visual representation of all possible product structures, they are not efficient in integrating functional requirements of products [13].

A Design Structure Matrix (DSM) is another common technique in the design literature which is used to enhance a designer's understanding of the architecture of the system and support decision making in redesigning the product. DSM has been used for modeling product decomposition, structure analysis, and identification of the interfaces among parts in a product [14–16].

In this paper, we will use coloring and unification of graphs to represent changing requirements during different iterations of the product. The final coloring of the unified graph can help designers implement and manage design changes with greater accuracy, and furthermore, can help define the proper number of physical parts that to cover functional requirements over several product iterations.

PROPOSED GRAPH COLORING METHOD

The proposed method addresses three needs of the designers: (1) designers understand the effects of adding or removing functional requirements during subsequent iterations, (2) they can define which functional requirements remain fixed during the product's lifespan based on the service agreements, and (3) they can define which functional requirements have the most potential for ongoing alteration.

To model these relations we can use graphs. A graph G is composed of a set of nodes called the vertex set $V(G)$ and an edge set defined as $E(G) = \{vw \mid vw \text{ is an edge connecting nodes } v \text{ and } w\}$. A subgraph $S \subseteq G$ is defined as any subset of vertices of G with edges between them in $E(G)$. Independent sets within G are defined as a set of vertices $I \subseteq G$ such that $I = \{v_i \mid v_i v_j \text{ is not an edge in } E(G)\ \forall\ i, j\}$. An independent set I is *maximal* if it cannot be extended by the inclusion of any vertex $v \in G \setminus I$. Our preferred coloring algorithm takes advantage of maximal independent sets within G to shorten the coloring process.

Step 1: Construct Input Graphs

The method begins with N graphs $\{G_1, G_2, G_3, ..., G_N\}$ created by the designer to represent N iterations of a product. Each graph G_i is made up of:

$V(G_i)$: a set of vertices, where each vertex represents a Functional Requirement (FR) of the product during the *i*th iteration.
$E(G_i)$: a set of edges where each edge represents a fundamental conflict between FRs during the *i*th iteration.

All vertices and edges are set by the designer. Note that the designer may choose to add Functional Requirements (as vertices) to successive G_i's, and/or new edges representing conflicts. For example, in order for a product to be easily repaired, it may need an additional feature that conflicts with one of the functional requirements of the original design. Alternatively, the designer may choose to remove Functional Requirements and/or edges in subsequent iterations of a product - i.e., certain features and requirements of the design may become obsolete in future iterations. See Example 2, below, for a demonstration of how Functional Requirements may shift between iterations.

The designer places edges between vertices (FRs) that should not be combined into a single part of the finished product. For example, in the design of a mechanical pencil, the FR of 'marking' and the FR of 'erasing' have the fundamental conflict that they cannot be the same material. A designer would place an edge between the vertices representing these FRs in the graph representing the pencil.

Furthermore, the designer defines a set of labels $k \in \{1,2,3,...\}$, where each label k corresponds to a subset of vertices in G_i. These labels are used to group vertices according to any design specification of interest. Examples of possible uses include grouping FRs with similar expiry dates, or those with equivalent costs to replace. These subsets of vertices grouped by label partition $V(G_i)$, so that each vertex has one and only one label. The information in these labels will allow us to increase the efficiency of our coloring algorithm, in a method inspired by the work of Eppstein [17] and Byskov [18] which improved coloring algorithms by looking at maximal independent subsets of graphs.

The completed iteration graphs G_i are the inputs to Algorithm 1.

Step 2: Define Core Graph

The next task is to identify the *core graph*, G_C which will include those Functional Requirements common to all N graphs. We build the vertex set of G_C by taking the intersection of all $v(G_i)$ for each $G_i \in \{G_1, G_2, ..., G_N\}$, as in Eqn. 1:

$$\bigcap_{i \leq n} V(G_i) = V(G_C) \tag{1}$$

We build the edge set of G_C by first finding a subgraph $H_{iC} \in G_i$ for each i such that H_{iC} includes all edges induced by $V(G_C)$. We then take the union of edges $E(H_{iC})$ in those subgraphs as in

Input: A set of graphs $G_1, G_2, ..., G_N$
Output: Void
Result: A set of N properly-colored graphs
1 Let G_C be an empty graph;
2 **for** *each vertex* $u \in V(G_1 \cap G_2 \cap ... \cap G_N)$ **do**
3 | $G_C = G_C + u$;
4 **end**
5 **for** *each edge uv in* G_C **do**
6 | **if** $e \in E(G_1 \cup G_2 \cup ... \cup G_N)$ **then**
7 | | $G_C = G_C + uv$;
8 | **end**
9 **end**
10 Let c = 1;
11 c = colorByLabel(c, G_c);
12 **for** *each* G_i *of* $G_1, G_2, ..., G_N$ **do**
13 | G_i=$G_i - G_C$;
14 | colorByLabel(c, G_i);
15 | $G_i = G_i \cup G_C$;
16 **end**

Algorithm 1: main

Input: A graph H, each vertex having a numeric label
A number c, the next available color
Output: A number c representing the last used color
Result: H is now properly colored
1 **for** *each label L in H* **do**
2 | Let H_L be the subgraph of vertices of H having label L;
3 | c = color(H_L);
4 **end**
5 return c;

Algorithm 2: colorByLabel

Eqn. 2:

$$\bigcup_{i \leq n} E(H_{iC}) = E(G_C) \tag{2}$$

This process occurs during lines 1-9 of Algorithm 1.

Step 3: Color Core Graph

Now that G_C has been identified, it can be colored.

(3)(a) Beginning with color $c = 1$ and the smallest natural number label k present in G_C, we properly color all those vertices with label k using any convenient coloring algorithm. We prefer the method enumerated in [18]. This coloring algorithm finds the chromatic number of G by populating an array X of length 2^n

with the chromatic numbers of all the subgraphs in the power set of G, including G itself. It then colors G by finding a subgraph $S \subset G$ such that $X[S] = X[G] - 1$, and then assigns the first color to the vertices of $G - S$. It then finds a subgraph $T \subset S$ such that $X[T] = X[S] - 1$ and assigns color 2 to the vertices of $S - T$ and so on until all the vertices of G are colored. Note that after we have applied the coloring algorithm that the last color used in this process is currently stored in the variable c.

(3)(b) Once all vertices with label k are colored, we move to label $k+1$, and increment to the next color, $c = c+1$. This ensures that parts with different labels are guaranteed to have different colors in the final graph. Once we have made it through every label, the core graph G_C is properly colored, and the number $c+1$ contains the next available color. This occurs during lines 10-11 of Algorithm 1, which invokes Algorithm 2.

Step 4: Color Remaining Vertices

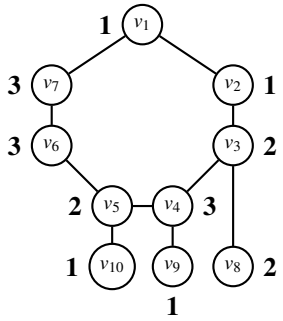
The next step is to color those vertices of each G_i in $\{G_1, G_2, ..., G_N\}$ that are outside of G_C. Note that we'll be starting with next available color after those in G_C. We find $G_i \setminus G_C$, which is all vertices and edges in G_i that do not appear in G_C. Then as before, we color $G_i \setminus G_C$ by label as in Algorithm 2. We then build the unified graph $(G_i \setminus G_C) \bigcup G_C$, which is a properly colored copy of G_i. This occurs over lines 12-16 of Algorithm 1. Note that we begin the complete coloring of subsequent iterations $G_j \setminus G_C$ where $j > i$ with the same initial color, the next available color after coloring the Core Graph.

ALGORITHM ANALYSIS

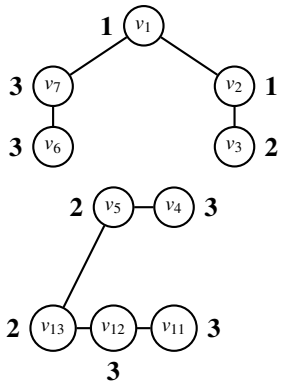
Per Byskov's improvement of Epstein's coloring algorithm, it takes $O(2.0423^n + 2^n)$ time to properly color a general graph of n nodes [18] [17]. In our algorithm, we have N graphs to color, raising the time to $O(N(2.0423^n + 2^n))$. The fact that we are coloring by label, however, reduces this complexity somewhat. Supposing that each graph has k labels and that the number of vertices under each label is about n/k, then for each graph G_i we are coloring k subgraphs of G_i with n/k nodes in the subgraph. Our overall runtime is then $O(Nk(2.0423^{n/k} + 2^{n/k}))$. This compares favorably with minimum coloring runtimes established by Beigel and Lawler [19, 20].

EXAMPLES

We here include two examples of the coloring and unification process. The first example represents an abstract product with two input graphs. The second example is a real-world example showing three iterations of a tethered Wi-Fi and Bluetooth enabled speaker, which has three input graphs. Each input graph represents some iteration of the product. In each example, labels



(a) Ex. 1 G_1



(b) Ex. 1 G_2

FIGURE 1: EXAMPLE 1 INPUT GRAPHS WITH LABELS

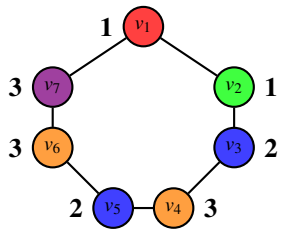
are shown as boldface numbers beside each node. Each node represents a specific FR, and the labels represent some shared quality such that we would like to unite those FRs into a single part if possible. This desire is reflected in the algorithm itself, which breaks the input graphs down label-wise, and tries to color nodes of the same label with the same color, only failing to do so if the presence of an edge forbids it.

Example 1

Step 1 Consider the example input graphs in Fig. 1.

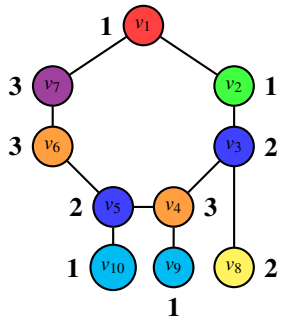
Step 2 From these input graphs G_1 and G_2, we take the intersection of vertices appearing in both graphs to find the core graph G_C.

Step 3a Once G_C is found, we begin to color the graph by first examining and coloring the vertices labelled **1**. Thus, any vertex labelled **1** will receive the same color, unless an edge between vertices (representing a conflict between functional requirements) forbids this. In this case, since the two vertices labelled **1** are adjacent to one another, we color the first, v_1, red,

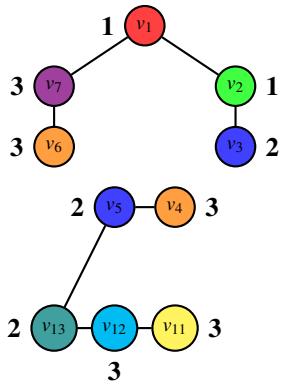


(a) Ex. 1 G_C

FIGURE 2: EXAMPLE 1 CORE GRAPH WITH COLORING



(a) Ex. 1 Result G_1



(b) Ex. 1 Result G_2

FIGURE 3: EXAMPLE 1 COLORING RESULT

and the second, v_2, green.

Step 3b Once all vertices labelled **1** are colored, we proceed to vertices labelled **2**. At this stage, we choose the next available color not used on vertices of label **1**, thus guaranteeing that vertices of different labels will have different colors. In this case, the two vertices labelled **2** are not adjacent, so they are both colored blue. This process continues through all the labels until G_C is properly colored (see Fig. 2).

Step 4 Once the core graph is colored, we return to each of the input graphs G_1 and G_2 and color remaining vertices, again proceeding by label, and beginning each G_i with the next available color after coloring the Core Graph. A final coloring of the input graphs is shown in Fig. 3. The graph G_1 has been colored with 7 colors and the graph G_2 with 8 colors. Thus the item represented by these graphs can be manufactured with 7 or 8 colors depending on which input graph is of more importance to the designer. If the designer chooses to retain the complete functional requirements of both iterations, they will need 10 distinct parts because 10 colors are used to color both of the two input graphs completely.

Example 2

The second example is a real-world product: an internet-enabled speaker. In our hypothetical scenario, the designer is considering three iterations of the product:

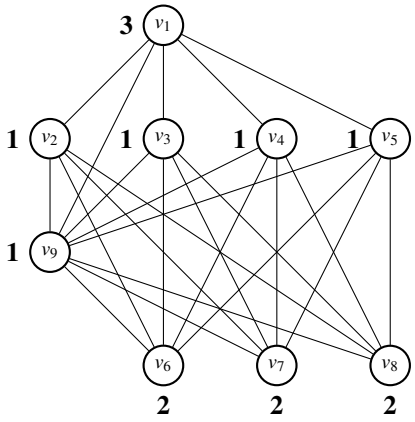
1. **Basic Model:** The product has modest performance specifications and is controlled via Bluetooth only.
2. **Voice Assistant Model:** The product has modest performance specifications and is Bluetooth and Wi-Fi enabled, and has a microphone for voice control.
3. **Hi-Fi Model:** The product is Bluetooth and Wi-Fi enabled, has a microphone for voice control, uses a high performance speaker and amplifier, and has manual volume control.

The designer will apply our coloring algorithm to fix which functional requirements will appear in each iteration, and to determine the effects of these differences on the number of parts required. Additionally, application of the algorithm will give the designer an idea of which functional requirements have the capacity for ongoing development throughout the product's life cycle.

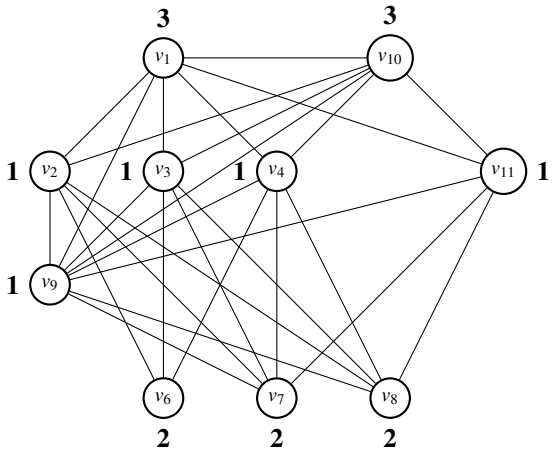
Step 1 Consider the example input graphs in Fig. 4. Each vertex, in addition to its index, is assigned a label (appearing here as a bold numeral next to the vertex). Any number of labels can be used, and these labels could represent any classifications the designer wishes to apply to the various functional requirements. Possible uses for the labels include distinguishing between FRs with different development or upgrade cycles; FRs designed by different departments in a large corporation; or FRs with different software requirements. In this case, let there be three labels representing supply chain sources for each type of part, roughly classified as **1**: electronic, **2**: physical, and **3**: acoustic. Table 1 specifies the assignment of functional requirements to labels and vertices, and indicates which FRs are included in each iteration.

TABLE 1: EXAMPLE 2: TETHERED SPEAKER FUNCTIONAL REQUIREMENT VERTEX ASSIGNMENT

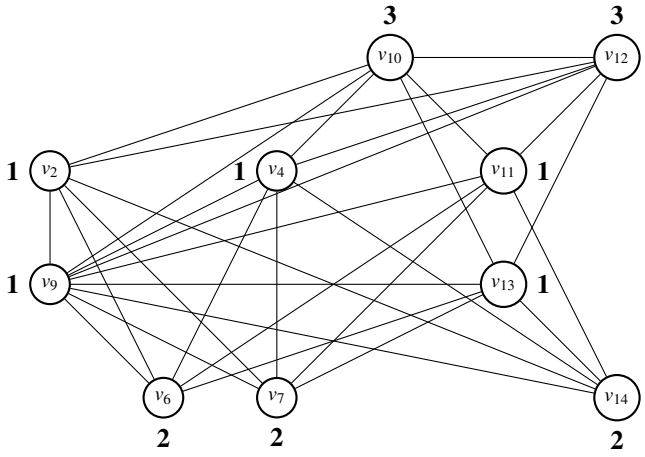
			Iterations		
Functional Requirement	**Vertex**	**Label**	**Basic Model**	**Voice Assistant**	**Hi-Fi Model**
Sound Output	v_1	3	✓	✓	
Improved Sound Output	v_{12}	3			✓
Sound Input	v_{10}	3		✓	✓
Power Input	v_2	1	✓	✓	✓
Amplification	v_3	1	✓	✓	
Improved Amplification	v_{13}	1			✓
Computation	v_4	1	✓	✓	✓
Bluetooth	v_5	1	✓		
Bluetooth & Wi-Fi	v_{11}	1		✓	✓
Enclosure	v_6	2	✓	✓	✓
Manual Level Control	v_{14}	2			✓
Activation	v_7	2	✓	✓	✓
Portability	v_8	2	✓	✓	
Power Storage	v_9	1	✓	✓	✓



(a) Ex. 2 G_1 Basic Model



(b) Ex. 2 G_2 Voice Assistant Model



(c) Ex. 2 G_3 Hi-Fi Model

FIGURE 4: EXAMPLE 2 INPUT GRAPHS WITH LABELS

Step 2 Following the unification process carried out by Algorithm 1, the Core Graph for these inputs is shown in Fig. 5. Note that the Core Graph contains only the following Functional Requirements: v_2 Power Input, v_4 Computation, v_9 Power Storage, v_6 Enclosure, and v_7 Activation. These are the functional requirements which are common to each iteration of the speaker.

Step 3(a,b) We color G_C by label with a proper coloring according to Algorithm 2. We first color all vertices labelled **1**. In this case, there is no conflict between v_2 (Power Input) and v_4 (Computation), so these two vertices can both receive the color red. Since the remaining vertex of label **1**, v_9 (Power Storage), has a conflict with the other two vertices labelled **1**, it gets the next available color, green. Note: this conflict stems from the impracticability of affixing a large, removable battery to the same circuit board as the parts associated with the other Functional Requirements labelled **1**. Next, we move on to label **2**, and since there is no conflict between v_6 (Enclosure) and v_7 (Activation), we assign the next available color, blue, to both vertices labelled **2**. Thus we see that three colors are sufficient to achieve a proper coloring of the core graph G_C. This indicates that three basic components, corresponding to the three colors, can be built and used in the assembly of all three iterations:

G_C **Core Graph Coloring Interpreted as Parts**

Red Single circuit board with power supply (Power Input) and system on a chip (Computation)

Blue Case (Enclosure) with built-in on/off button (Activation)

Green Replaceable battery (Power Storage)

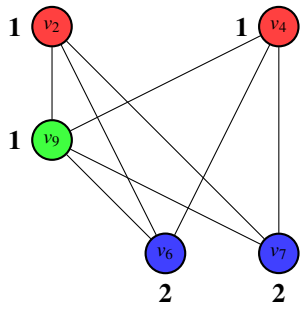
Step 4 Once the Core Graph is colored, we return to the remaining uncolored vertices in each iteration. We begin coloring $G_1 \setminus G_C$ with the next available color after core coloring is complete: $c+1$ (in this case, orange) in reference to the output c of Algorithm 1. We then proceed to color by label as in Algorithm 2, beginning with vertices labelled **1**. Since v_3 (Basic Amplification) and v_5 (Bluetooth) have no conflict, they can both be colored orange. We proceed to v_8 (Portability), the only uncolored vertex labelled **2** in this iteration, which receives the next available color: purple. Finally, we color the lone vertex labelled **3**, v_1 (Basic Sound Output) yellow. Six colors suffice to completely color G_1, indicating that the Functional Requirements could be combined into six independent parts when the Basic Model of the speaker is built (see Figure 7a for a rendering):

G_1 **Basic Model Coloring Interpreted as Parts**

Red Single circuit board with power supply (Power Input) and system on a chip (Computation)

Blue Case (Enclosure) with built-in on/off button (Activation)

Green Replaceable battery (Power Storage)



(a) Ex. 2 G_C Core Graph

FIGURE 5: EXAMPLE 2 CORE GRAPH WITH COLORING

Orange Circuit board with a 10 Watt 10db S/N Amplifier (Basic Amplification) and a bluetooth antennae (Bluetooth Connectivity)

Purple Handle (Portability)

Yellow 5 Watt Speaker (Basic Sound Output)

We proceed next to $G_2 \setminus G_C$, which we will color by label as in Algorithm 2. Note that we will begin coloring $G_2 \setminus G_C$ with the same initial color that we used to begin coloring $G_1 \setminus G_C$: orange. Since the two vertices labelled **1**, v_3 (Basic Amplification) and v_{11} (Bluetooth and Wi-Fi), have no conflict, they are both labelled orange. The lone uncolored vertex labelled **2**, v_8 (Portability), is colored purple. Finally, the two vertices labelled **3** have a conflict in the form of a shared edge, so they are colored yellow for v_1 (Basic Sound Output) and light blue for v_{10} (Sound Input). Thus seven colors complete the coloring of G_2, indicating that the Functional Requirements could be combined into seven independent parts when the Voice Assistant Model of the speaker is built (see Figure 7c for a rendering):

G_2 **Voice Assistant Model Coloring Interpreted as Parts**

Red Single circuit board with power supply (Power Input) and system on a chip (Computation)

Blue Case (Enclosure) with built-in on/off button (Activation)

Green Replaceable battery (Power Storage)

Orange Circuit board with a 10 Watt 10db S/N Amplifier (Basic Amplification) and a combined Bluetooth and Wi-Fi antennae (Bluetooth and Wi-Fi)

Purple Handle (Portability)

Yellow 5 Watt Speaker (Basic Sound Output)

Light Blue Microphone (Sound Input)

Finally, we begin to color $G_3 \setminus G_C$, again starting with orange as the first available color after core coloring is complete. This iteration of the product includes the FRs of Improved Sound Output and Improved Amplification, so will include new vertices. Since the two vertices labelled **1**, v_{13} (Improved Amplification)

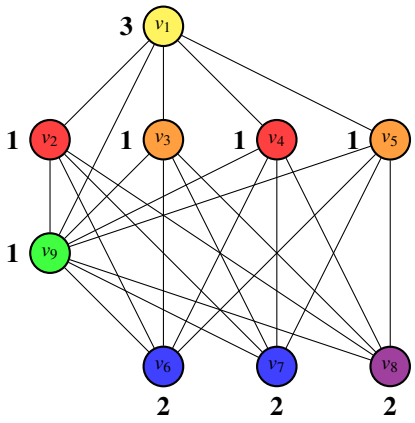
and v_{11} (Bluetooth and Wi-Fi), have no conflict, they are both labelled orange. The lone uncolored vertex labelled **2**, v_{14} (Manual Level Control), is colored purple. Finally, the two vertices labelled **3** have a conflict in the form of a shared edge, so they are colored yellow for v_{10} (Sound Input) and light blue for v_{12} (Improved Sound Output). Thus seven colors complete the coloring of G_3, indicating that the Functional Requirements could be combined into seven independent parts when the Hi-Fi Model of the speaker is built (see Figure **??** for a rendering):

G_3 Hi-Fi Model Coloring Interpreted as Parts

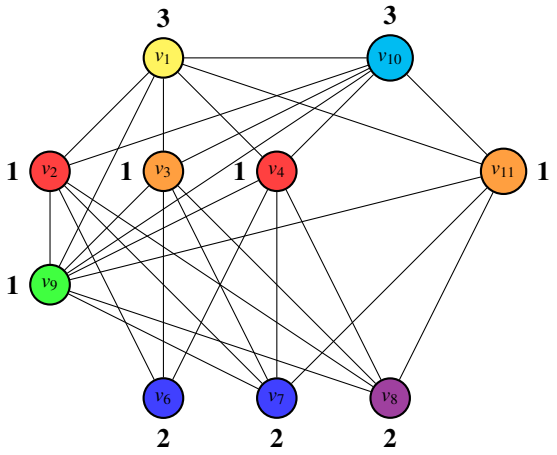
Red	Single circuit board with power supply (Power Input) and system on a chip (Computation)
Blue	Case (Enclosure) with built-in on/off button (Activation)
Green	Replaceable battery (Power Storage)
Orange	Circuit board with a 100 Watt 50 db S/N Amplifier (Improved Amplification) and a combined Bluetooth and Wi-Fi antennae (Bluetooth and Wi-Fi)
Purple	Volume Buttons (Manual Level Control)
Yellow	Microphone (Sound Input)
Light Blue	50 Watt Speaker (Improved Sound Output)

A final coloring of the input graphs is shown in Figure 6. A rendering of possible build structures for the three iterations is shown in Figure 7. For each iteration, every color in the final coloring has been interpreted as a part with one or more components. These parts are outlined in the color used in the corresponding graph for each iteration. Note that this technique allows the designer to clearly see which functional requirements remain fixed over time, and which can be modified over time. Additionally, the changes to the product over time, as FR's are added and removed, is clearly rendered in both the coloring graphs and the final renderings.

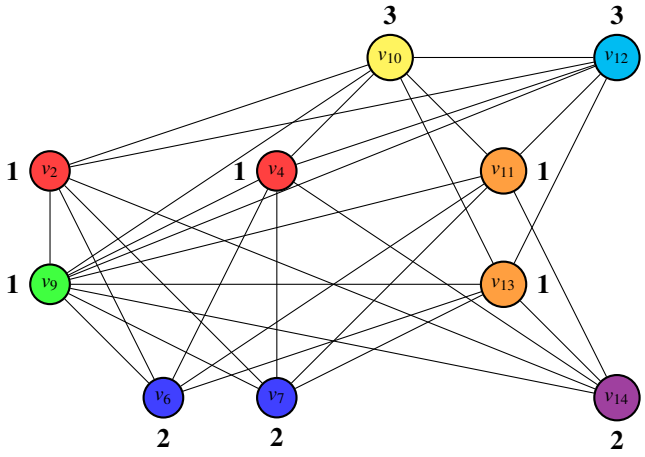
This is a small-scale example of how our proposed coloring method can be used to design for multiple product iterations, and as such is quite simple. We include this example to show how the method works, and to indicate the utility of our proposed algorithm in cases with many more iterations and Functional Requirements.



(a) Ex. 2 G_1 Basic Model Final Coloring



(b) Ex. 2 G_2 Voice Assistant Model Final Coloring



(c) Ex. 2 G_3 Hi-Fi Model Final Coloring

FIGURE 6: EXAMPLE 2 COLORING RESULT

CONCLUSION

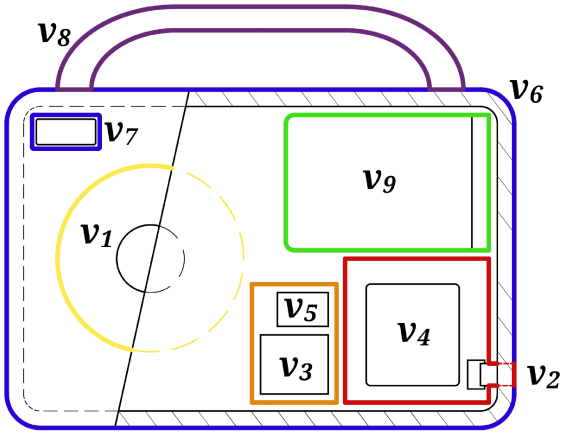
The combination of embedded software and smart devices has fundamentally changed the way products and services are offered to customers. New business models are emerging that blur the edge between service and product, alter product ownership among different stakeholders, and change system functionalities over time. These changes in the market resulting from consumer behavior, technology advancement as well as marketing strategies adopted by businesses require a new set of design methods that increase product lifetime value. Considering the dynamic nature of the market, designers should be enabled to see the impacts of adding or revising product functional requirements which can improve their design decisions .

In this study, a graph unification method is suggested to combine different product iterations together under one core graph and further, a graph coloring algorithm is developed to determine the minimum number of parts needed to accommodate dynamic changes in product functionalities. The study opens the door for more research in the field of design for flexibility.

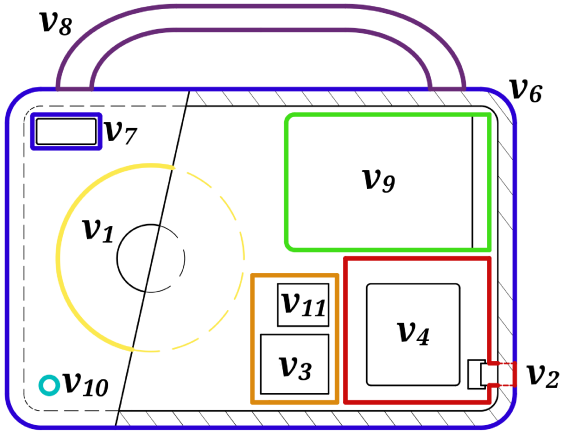
This flexibility is most clearly demonstrated in our second example of a IoT speaker with several design iterations. Our algorithm, when applied to the graph representations of the product, quickly identifies those FRs that are common and have consistent mathematical structure between iterations in the form of the Core Graph, and breaks those FRs into a minimal number of parts. This part set is common to each iteration, and any differentiation among iterations must build up from this core.

Once the core of the product is fixed, the characteristics of the different iterations give rise to sets of parts particular to those iterations. This demonstrates the flexibility of this approach, as new versions of the product can be easily designed by adding parts to the core in different ways. Future iterations beyond those used to compile the core graph may nonetheless use it as a starting point.

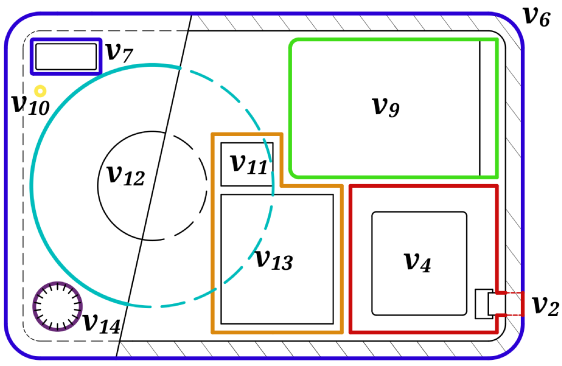
This study can be extended in several ways. First, the model can be applied to a real case of a smart device and its corresponding service contract. Second, graph coloring techniques can be augmented with uncertainty quantification models to further study the effects of uncertain consumer behavior, market changes, and technological advancement. Third, graph coloring algorithms can be integrated with economic models to quantify the business case of design decisions. Finally, the concept of design for flexibility can be integrated with the circular economy concept to study the sustainability impacts of flexible design.



(a) G_1: Basic Model Rendering



(b) G_2: Voice Assistant Model Rendering



(c) G_3: Hi-Fi Model Rendering

FIGURE 7: FINISHED PRODUCT RENDERINGS WITH COLORS INTERPRETED AS PARTS

ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation–USA under grants CMMI-2017968 and CMMI-1903810. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Zittrain, J., 2008. *The Future of the Internet - and How to Stop It*. Yale University Press, New Haven, CT.

[2] Hoofnagle, C., Kesari, A., and Perzanowski, A., 2019. "The tethered economy". *George Washington Law Review,* ***87***(4), July, p. 783.

[3] Gopalakrishnan, P., Cavallaro, J., Jahanbekam, S., and Behdad, S., 2019. "A graph coloring technique for identifying the minimum number of parts for physical integration in product design". In International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Vol. **4** of *American Society of Mechanical Engineers Digital Collection*. Paper number DETC2019-98251.

[4] Kiritsis, D., Bufardi, A., and Xirouchakis, P., 2003. "Research issues on product lifecycle management and information tracking using smart embedded systems". *Advanced Engineering Informatics,* ***17***(3), July, pp. 189–202.

[5] Kjaer, L., Pigosso, D., Niero, M., Bech, N., and McAloone, T., 2019. "Product/service systems for a circular economy: The route to decoupling economic growth from resource consumption?". *Journal of Industrial Ecology,* ***23***(1), February, pp. 22–35.

[6] Lewandowski, M., 2016. "Designing the buisness models for circular economy - towards the conceptual framework". *Sustainability,* ***8***(1), January, p. 43.

[7] Svensson, S., Richter, J., Maitre-Ekern, E., Pihlajarinne, R., Maigret, A., and Dalhammar, C., 2018. The emerging 'right to repair' legislation in the eu and the u.s. Paper Presented at Going Green CARE INNOVATION 2018 Conference, November.

[8] Crootof, R., 2019. "The internet of torts: Expanding civil liability standards to address corporate remote interference". *Duke Law Journal,* ***69***(3), December, pp. 583–667.

[9] Zhang, G., Morris, E., Allaire, D., and McAdams, D., 2020. "Research opportunities and challenges in engineering system evolution". *Journal of Mechanical Design,* ***142***(8), August.

[10] Vinodh, S., and Rathod, G., 2014. "Application of life cycle assessment and monte carlo simulation for enabling sustainable product design". *Journal of Engineering, Design and Technology,* ***12***(3).

[11] Zhao, Y., Xu, J., and Thurston, D., 2011. "A hierarchical bayesian method for market positioning in environmentally conscious design". In Proceedings of the ASME Design Engineering Technical Conference, Vol. **9** of *American Society of Mechanical Engineers Digital Collection*, pp. 3–16. Paper number DETC2011-47898.

[12] Du, X., 2006. "Uncertainty analysis with probability and evidence theories". In Proceedings of the ASME Design Engineering Technical Conference, Vol. **1** of *American Society of Mechanical Engineers Digital Collection*, pp. 1025–1038. Paper number DETC2006-99078.

[13] Behera, A., McKay, A., Earl, C., Chau, H., Robinson, M. A., de Pennington, A., and Hogg, D., 2019. "Sharing design definitions across product life cycles". *Research in Engineering Design,* ***30***(3), January, pp. 339–361.

[14] Tilstra, A., Seepersad, C., and Wood, K., 2009. "Analysis of product flexibility for future evolution based on design guidelines and a high-definition design structure matrix". In Proceedings of the ASME Design Engineering Technical Conference, Vol. **5** of *American Society of Mechanical Engineers Digital Collection*, pp. 951–964. Paper number DETC2009-87118.

[15] Eppinger, S., and Browning, T., 2012. *Design Structure Matrix Methods and Applications*. MIT Press, Cambridge, MA.

[16] Borjesson, F., and Hölttä-Otto, K., 2012. "Improved clustering algorithm for design structure matrix". In Proceedings of the ASME Design Engineering Technical Conference, Vol. **3** of *American Society of Mechanical Engineers Digital Collection*, pp. 921–930. Paper number DETC2012-70076.

[17] Eppstein, D., 2003. "Small maximal independent sets and faster exact graph coloring". *Journal of Graph Algorithms and Applications,* ***7***(2), pp. 131–140.

[18] Byskov, J. M., 2004. "Enumerating maximal independent sets with applications to graph colouring". *Operations Research Letters,* ***32***, November, pp. 547–556.

[19] Beigel, R., and Eppstein, D., 2005. "3-coloring in time $\mathcal{O}(1.3289^n)$". *Journal of Algorithms,* ***54***(2), February, pp. 168–204.

[20] Lawler, E., 1976. "A note on the complexity of the chromatic number problem". *Information Processing Letters,* ***5***(3), August, pp. 66–67.