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

IDETC2020-22774

ENHANCEMENTS TO THE PERFECT MATCHING APPROACH FOR GRAPH ENUMERATION-BASED ENGINEERING CHALLENGES

Daniel R. Herber
Colorado State University
Department of Systems Engineering
Fort Collins, CO 80523
Email: daniel.herber@colostate.edu

ABSTRACT

Graphs can be used to represent many engineering systems and decisions because of their ability to capture discrete compositional and relational information. In this article, improved methods for effectively representing and generating all graphs in a space defined by certain complex specifications are presented. These improvements are realized through enhancements to the original perfect matching-inspired approach utilizing a component catalog definition to capture the graphs of interest. These enhancements will come in many forms, including more efficient graph enumeration and labeled graph isomorphism checking, expansion of the definition of the component catalog, and the effective inclusion of new network structure constraints. Several examples are shown, including improvements to the original case studies (with up to 971× reduction in computational cost) as well as graph problems in common system architecture design patterns. The goal is to show that the work presented here and tools developed from it can play a role as the domain-independent architecture decision support tool for a variety of graph enumeration-based engineering design challenges.

1 INTRODUCTION

Graphs are one of the fundamental objects in mathematics but are also essential to many fields in science and engineering because graphs can capture discrete compositional and relational information through vertices and edges. Many studies utilize graph representations and generation methods including problems in areas such as general system architecture decisions [1, 2], automotive [3–8], electric circuits [9, 10], aircraft [11–13], geartrains [14–16], chemical molecules [17–19], biochemical topologies [20], and others [21–26]. Graph enumeration, or the generation of all graphs that satisfy some certain set of specifications, has played a role in many of these studies [1–7, 9, 11, 12, 14, 16, 18–20, 23, 25–28].

There is a variety of constructions that can formalize the graphs of interest. These constructions create graphs from a potentially constrained set of graph-modifying actions such as adjacency/connectivity matrices [1, 5, 29], rules and graph grammars [15, 22, 30–33], and catalogs [3, 4, 6, 23, 26–28, 34]. Many of these representations are used within an optimization framework [7, 8, 10, 13, 15, 21, 25, 33, 35] to identify the best or most promising candidate solutions [11, 13, 24, 36, 37]. In this work, we will only consider the task of generating all graphs. While enumeration-based methods will always suffer from combinatorial complexity [1, 36], the development of both the theory and tools for generating graphs based on complex specifications is possible for many pressing engineering design challenges.

The basis of this work is the catalog-based, perfect matching (PM)-inspired algorithm presented in Ref. [34] for generating a desired set of graphs. This approach was shown to have a certain level of success over alternative enumeration strategies by limiting the number of generated infeasible and nonunique (nonisomorphic) graphs. Some infeasible graphs were avoided by directly including some network structure constraints (NSCs) [11, 36, 37] in the generation process. Nonunique graphs were avoided by leveraging the assumed structure of the graphs of interest. However, many improvements to this ap-

ALGORITHM 1: Original perfect matching-inspired, recursive, brute-force algorithm.

Input : V – vector of remaining ports for each component replicate
E – vector of edges in sequential pairs (initially empty)
A – expanded potential adjacency matrix
cVf – cumulative sum of the original V plus 1
G – set of graphs, initially empty
Output: G – set of graphs

```
1  iL ← find(V,first)                          // find first nonzero entry
2  L  ← cVf(iL) – V(iL)                                  // left port
3  V(iL) ← V(iL) – 1                                    // remove port
4  Vallow ← V ∘ A(iL,:)                        // zero infeasible edges
5  I  ← find(Vallow)                           // find nonzero entries
6  for iR ← I do                       // loop through all nonzero entries
7      R  ← cVf(iR) – V(iR)                              // right port
8      E2 ← [E,L,R]          // combine left, right ports for an edge
9      V2 ← V                          // local remaining ports vector
10     V2(iR) ← V2(iR) – 1                 // remove port (local copy)
11     A2 ← A           // local expanded potential adjacency matrix
12     if all V2 is zero then                  // no remaining connections
13         G{end + 1} ← E2          // save missorted perfect matching
14     else
15         G ← Algorithm 1 with (V2,E2,A2,cVf,G)          // recursive
16     end
17 end
18 G ← Convert all G to a proper graph
```

proach are possible.

1.1 Original Algorithm

In Ref. [34], a PM-inspired, brute-force algorithm was presented that generates the set of vertex-labeled graphs in the graph structure space $\mathcal{G}$ [29,34] defined by $(\boldsymbol{L},\boldsymbol{P},\boldsymbol{R})$ and various additional NSCs [34,36]. The collection $(\boldsymbol{L},\boldsymbol{P},\boldsymbol{R})$ is termed the *component catalog* where: $\boldsymbol{L}$ is the label sequence representing n component types[1], $\boldsymbol{P} \in \mathbb{N}_0^n$ is the port sequence indicating the number of ports (or connections) for each component type, and $\boldsymbol{R} \in \mathbb{N}^n$ is the replicate sequence indicating the number of replicates for each component type. The $(\boldsymbol{L},\boldsymbol{P},\boldsymbol{R})$ catalog representation can be alternatively expressed as:

$$\boldsymbol{D}^L = \left[(P_1)^{L_1}, \overset{\times R_1}{\cdots}, (P_1)^{L_1}, \cdots, (P_n)^{L_n}, \overset{\times R_n}{\cdots}, (P_n)^{L_n}\right] \tag{1}$$

which is termed the labeled expanded port sequence and can also be canonically ordered for a unique representation of a particular component catalog. We also denote the total number of ports and replicates as $N_p = \boldsymbol{P} \cdot \boldsymbol{R}$ and $N_r = \boldsymbol{1} \cdot \boldsymbol{R}$, respectively. Finally, the expanded port sequence $\boldsymbol{D}$ is simply $\boldsymbol{D}^L$ without the label superscripts and is in $\mathbb{N}_0^{N_r}$. An example is shown in Fig. 4.

The original graph enumeration algorithm is shown in Alg. 1. This is a PM-inspired algorithm because the algorithm seeks to generate all valid PMs, where a PM is a set of edges such that no two have a vertex in common and all vertices contain exactly one edge [38]. Therefore, the upper bound on the number of graphs generated is $(N_p - 1)!!$ [34, 38] or A001147 [39]. Unfortunately, the generated set of graphs G can be (sometimes much) larger than $\mathcal{G}$ because $\mathcal{G}$ is defined as all unique feasible graphs (UFGs) while Alg. 1 can produce graphs which do not satisfy all specified NSCs and are isomorphic to another in G (see Definition 1). One of the motivating factors behind the structure of Alg. 1 was specific features that efficiently avoid producing infeasible, isomorphic graphs during the generation process.

Only a few additional NSCs were directly addressed in Ref. [34], namely connected graphs, mandatory components (i.e., must be in the final graph), path constraints (i.e., two types must or must not have a path [40, p. 6] between them), multiedges, and direct connection constraints (i.e., disallowing particular types to be connected). Only direct connection constraints were handled effectively in Alg. 1; all graphs in G are feasible with respect to any direct connection constraints because of the operations on line 4. The others, including any additional NSCs, were checked for each candidate graph once the algorithm terminated.

As previously mentioned, there are a few other catalog-based approaches for defining $\mathcal{G}$ [3, 6, 23, 26–28]. Refs. [3, 28], in particular, use quite similar ideas for generating all graphs. Another popular technique is constructing graphs through constraint programming [4, 6, 26] approaches which can utilize efficient solvers. However, these methods can suffer isomorphism issues [6] and can be less effective than a tailored algorithm.

In the context of graph algorithms, Alg. 1 is not an orderly algorithm [41]. In an orderly algorithm, the next list of sequences is constructed from a previous sequence and augmenting operator. Each candidate sequence then has some test applied to it that is independent of the other sequences to ensure that it is needed (i.e., is in a canonical form). Here the sequences represent sets of edges, and the augmenting operator is a valid PM edge addition. However, no canonicity is performed in Alg. 1. A common test is for graph isomorphisms, but the inclusion of such a test would not make this an orderly algorithm because the test requires comparisons to all other sequences. The graph enumeration problem is also similar to the degree-constrained enumeration but with the addition of vertex labels.

1.2 Overview

In this article, we will describe several enhancements to the original algorithm. These enhancements will come in many forms, including more efficient graph enumeration, expanding the definition of the component catalog, and the effective inclusion of new NSCs.

The remainder of this article is organized as follows. Section 2 describes breaking the enumeration task into subcatalogs, while Sec. 3 discusses the various enhancements for the enumer-

[1]The original catalog definition in Ref. [34] used *C* and *colored* to indicate component types, but here we now prefer *L* and *labeled*, primarily because coloring in the context of graph theory typically implies that vertices with same color are not incident.

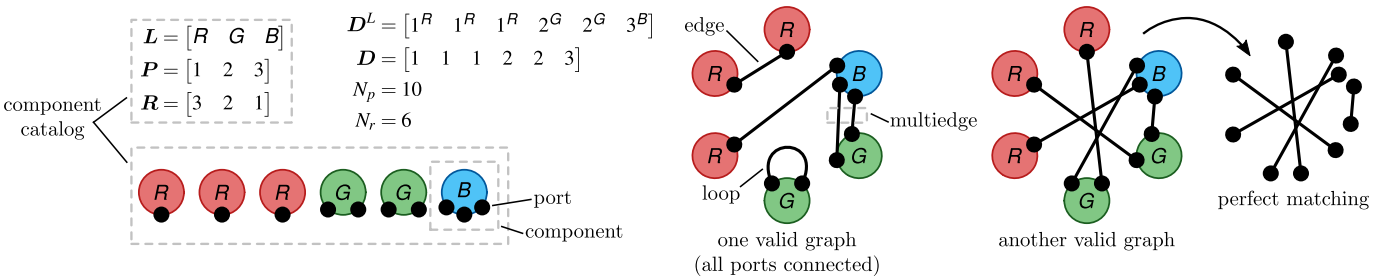


FIGURE 1: Example of the original catalog definition (catalog from Case Study 1).

ation of a single catalog. Section 4 describes the improvements checking for labeled graph isomorphisms in a set of graphs. Finally, Sec. 5 demonstrates the enhancements through several examples, and Sec. 6 presents the conclusions.

2 SUBCATALOGS IN THE GENERATION PROCESS

2.1 New Component Catalog Definition

Utilizing the original algorithm, G contained all graphs for every valid subcatalog of $(\boldsymbol{L}, \boldsymbol{P}, \boldsymbol{R})$ (i.e., graphs where the number of replicates for all components are bounded by $\mathbf{0}$ and $\boldsymbol{R}$) due to a property of enumerating PMs (namely, the edge set for problem size N_p contains all edge sets for $N_p - 2$). If we were interested in these subcatalogs, then unnecessary replicates could be removed [34]. However, this approach is decidedly inefficient and unintuitive. Here we consider the following more natural representation of the component catalog:

1. $\boldsymbol{L}$ is the label sequence representing n distinct component types (same as the previous definition)
2. $\boldsymbol{P} = [\underline{\boldsymbol{P}}, \bar{\boldsymbol{P}}]$ is the collection of two sequences that define lower and upper bounds on the number of ports for each component type with conditions: $\underline{\boldsymbol{P}} \le \bar{\boldsymbol{P}}, \underline{\boldsymbol{P}} \in \mathbb{N}_0^n, \bar{\boldsymbol{P}} \in \mathbb{N}_0^n$
3. $\boldsymbol{R} = [\underline{\boldsymbol{R}}, \bar{\boldsymbol{R}}]$ is the collection of two sequences that define lower and upper bounds on the number of replicates for each component type with conditions: $\underline{\boldsymbol{R}} \le \bar{\boldsymbol{R}}, \underline{\boldsymbol{R}} \in \mathbb{N}_0^n, \bar{\boldsymbol{R}} \in \mathbb{N}^n$

The modifications to $\boldsymbol{P}$ permit a more natural specification of port count bounds, such as all components of a particular type must have between 3 and 8 ports. Components in physical system modeling [8, 24] and system architecture decisions [1] can have nonfixed port counts. Similarly, the modifications to $\boldsymbol{R}$ permit a more natural specification of replicate count bounds, such as every graph must have been between 2 and 4 replicates. While various "tricks" were available to represent graph structure spaces with port and replicate bounds, these tricks were inefficient and unintuitive. We also note that a subcatalog definition could allow for general sets defining $\boldsymbol{P}$ and $\boldsymbol{R}$ (e.g., we require $\{0, 2, 4\}$ replicates), but for simplicity, we present only the current definition of the catalog.

2.2 Subcatalog Enumeration Algorithm

We now define the set of all subcatalogs C of $(\boldsymbol{L}, \boldsymbol{P}, \boldsymbol{R})$, under the new definition, as all possible valid combinations. With no additional constraints, the number of subcatalogs is:

$$|C| = \prod_{i=1}^{|\boldsymbol{L}|} \left[\sum_{j=\underline{R}_i}^{\bar{R}_i} \binom{j + \bar{P}_i - \underline{P}_i}{j} \right] \tag{2}$$

where the binomial coefficient represents the number of ways that j identical objects can be placed into $\bar{P}_i - \underline{P}_i + 1$ labeled bins (a variation of the "stars and bars" problem [42]). This equation provides the general structure of an algorithm to enumerate C: 1) go through each component type, 2) then go through each possible replicate value, 3) now generate all valid binnings of the current replicates, 4) then construct the Cartesian product of these partitions with previous intermediate catalogs, and 5) repeat until all component types have been considered. Two important computational techniques utilized were flexible array preallocation and memoization of the function that generates all binnings because the same inputs are frequently provided and the function is relatively expensive. To see the implementation, please refer to Ref. [43].

2.3 Subcatalog Constraints

The enforcement of NSCs on all subcatalogs can help improve the usefulness of $\mathcal{G}$ and reduce overall computational expense. Here we consider two classes of subcatalog constraints: branching and filtering. The characteristic feature of branching-type constraints (BTCs) is that the constraints can be checked during subcatalog enumeration. On the other hand, filter-type constraints (FTCs) can only be checked when a complete subcatalog is available because the constraint condition is only valid when the subcatalog composition can no longer change. For example, consider a constraint on the minimum value of N_p for a particular subcatalog. Then a subcatalog can only be declared infeasible if there are no more component types left to add because one additional component type could lead to the satisfaction of the constraint (so an FTC). However, for this particular constraint, we can bound the maximum number of additional ports that could be added at any stage of the subcatalog enumeration procedure (sim-

ilar to a branch and bound algorithm). Therefore, we have a BTC if implemented in this form. BTCs can be more effective at reducing overall computational expense because these constraints reduce the number of FTC checks. In both cases, arbitrary subcatalog constraints may be defined which use the current information available. Some possible subcatalog constraints include:

- *Linear penalty constraints* (BTC) where $\boldsymbol{\rho} \cdot \boldsymbol{R} \leq \alpha$ and $\boldsymbol{\rho}$ defines the penalty for each replicate. Some common penalty functions include a maximum system cost or mass. Upper bounds on the total number of ports ($N_p \leq \bar{N}_p$) and replicates ($N_r \leq \bar{N}_r$) are also specific linear penalty constraints.
- *Conflicting component constraints* (BTC) where either type L_i or L_j are included, but not both, represented as $R_i \parallel R_j \leq 1$.
- *Linear satisfaction constraints* (BTC) where $\boldsymbol{\rho} \cdot \boldsymbol{R} \geq \alpha$ and $\boldsymbol{\rho}$ defines the benefit for each replicate. A common linear satisfaction constraint is linked to system requirements where $\boldsymbol{\rho}$ is boolean valued and 1 indicates that replicate can satisfy the requirement, while 0 indicates it cannot. Lower bounds on the total number of ports ($N_p \geq \underline{N}_p$) and replicates ($N_r \geq \underline{N}_r$) are also specific linear satisfaction constraints.
- *Paired component constraints* (FTC) where if type L_i is included, then so should at least one L_j, represented as $\neg R_i \parallel \neg R_j \geq 1$.
- *Even total ports condition* (FTC) because for any subcatalog, the N_p must be even. This is a necessary condition for a PM and was specified in the original algorithm [34].
- A *simple graph condition* (FTC) based on the application of the Erdős-Gallai theorem [44] can be used when the graph is required to have no multiedges or loops (called a simple graph [40, 45]). If the condition is satisfied, then at least one simple graph exists (with no additional NSCs).
- A *connected graph condition* (FTC) can be used if a graph is required to be connected and have no multiedges or loops. Since a tree graph has the fewest number of edges to realize a connected simple graph with N_r vertices, a simple lower bound condition must be satisfied:

$$N_p \geq 2(N_r - 1) \tag{3}$$

  Equality can be enforced if the graph is required to be a tree graph.
- A *bipartite graph condition* (FTC) can be used when $\boldsymbol{A}_r$ is used to define a bipartite graph (using block diagonal zero matrices) [45, p. 6]. First, the number of ports for both types must be equal and the Gale-Ryser theorem must be satisfied [46].

2.4 Overall Procedure

Now that every possible subcatalog is directly enumerated, we no longer want to remove any vertices from the generated graphs in order to preserve the specific $\boldsymbol{D}^L$. Otherwise, such graph modifications could produce graphs that are already in another subcatalog or are the result of an invalid subcatalog. Therefore, we require that every replicate is mandatory when enumerating with subcatalogs.

ALGORITHM 2: Generate the set of unique feasible graphs using subcatalogs.

Input : $(\boldsymbol{L}, \boldsymbol{P}, \boldsymbol{R})$ – component catalog under new representation
NSC – network structure constraints
Output: g – set of unique feasible graphs

```
C ← Create subcatalogs using (L, P, R) and all branching constraints
  // see Secs. 2.2 and 2.3
C ← Filter C with all filtering constraints            // see Sec. 2.3
N ← Count rows in C                                    // number of subcatalogs
for k ← 1 to N do in parallel
    [l, p, r] ← Extract subcatalog k from C
    nsc ← Modify NSC for only the included replicates in r
    g{k} ← Generate feasible graphs using (l, r, p, nsc) and all
      replicates are mandatory
    g{k} ← Determine set of unique (nonisomorphic) graphs in g{k}
end
G ← Catenate all g                                     // each set is independent
```

The overall procedure for generating the set of unique, feasible graphs using subcatalogs is shown in Alg. 2. Both the graph generation and eventual isomorphism checks for a specific subcatalog can be performed parallel because none of the generated graphs in a specific subcatalog have the same canonical $\boldsymbol{D}^L$ as any other subcatalog (which is a necessary condition for two graphs to be isomorphic, see Sec. 4). Individual subcatalog enumeration can take varying amounts of time so randomized ordering or sorting by decreasing N_p can help reduce the average computation time.

Algorithm 1 does not leverage parallelization during the graph generation procedure. A similar procedure was presented in Ref. [47] for Alg. 2 but without the methods for generating subcatalogs using the new catalog representation and all subcatalog constraints.

3 ENHANCEMENTS FOR THE ENUMERATION OF A SINGLE CATALOG

In this section, we will focus on some enhancements for the enumeration of a *single* catalog. A technical report was previously published discussing many of these enhancements [47]. However, numerous changes have been made, which will be discussed in the following sections.

3.1 Alternative Spanning Tree Traversals

Algorithm 1 can be interpreted as generating paths in a directed graph with a root vertex (empty graph) and target vertices (PMs that define G which contains all graphs in $\mathcal{G}$). The union of these paths is a spanning tree between the root and target vertices. The vertices and edges between the root and targets are based on the allowable edge pairs in Alg. 1 and any termination conditions such as the detection of a saturated subgraph. This representation is visualized in Fig. 2 (cf. example in Fig. 4). This following

enhancements are based on the fact that there are different ways to create and traverse an appropriate spanning tree while still ensuring that every graph in $\mathcal{G}$ is present in G.

The first observation is based on the difference between depth-first search (DFS) and breadth-first search (BFS) tree traversal [48, 49]. Algorithm 1 can be classified as a recursive DFS implementation. A BFS implementation has a distinct feature not possible with DFS. Namely, since the tree is traversed by level in BFS (where a level is equivalent to adding one edge), isomorphism checking can be performed at each level to ensure that only the unique graphs continue to be enumerated. This difference is visualized in Ref. [47]. However, isomorphism checking can be expensive (see Sec. 4), and since many paths may result in infeasible graphs, checking if they are unique is an unnecessary expense. Therefore, it has been observed that neither is the clear choice for all graph enumeration problems. Generally speaking, it seems that in problems with more NSCs, DFS is favorable, while the BFS implementation is more effective with fewer NSCs. Note that a BFS implementation does not add any new vertices to the spanning but might remove some.

These next two modifications can change the specific vertices present in the spanning tree. The first is the simple observation that the ordering of $(\boldsymbol{L}, \boldsymbol{P}, \boldsymbol{R})$ effects the number of graphs generated using Alg. 1. Consider a catalog with two distinct single-replicate components with 2 and 4 ports, respectively. If Alg. 1 is applied with the 2-port component connected first, then 3 graphs are generated. However, if the 4-port component is connected first, then 4 graphs are generated. It has been generally observed that the most effective sorting method is using $\boldsymbol{P}$ such that types with fewer ports are connected first.

The second modification is termed touched vertex promotion. During the enumeration process, when a vertex has at least one edge, it is termed "touched". Then V is modified by promoting touched vertices to the front of V such that iL always choses a touched vertex before an untouched one. This enhancement still covers the desired $\mathcal{G}$ because we are simply reordering (and not removing) the available connections. It has been observed that we want to 1) arrive at graphs that are infeasible and 2) select vertices that generate fewer branches (feasible edges) earlier in the generation process. This enhancement is a heuristic that tries to help with both of these observations. In most cases, the inclusion of this enhancement has reduced number of vertices visited in the spanning tree and the size of G.

3.2 Replicate Ordering

This enhancement is similar to the port-ordering feature that Alg. 1 was designed around. Now, we will eliminate some of the component-type isomorphisms [34] during the graph generation process.

Consider a single component type and its set of n replicates. Now, consider a single replicate numbered n and $n \neq 1$. During an iteration of Alg. 1, a single edge is added. If replicate $n-1$ still has no ports connected, then adding this edge to replicate n will produce a graph isomorphic to the graph created when adding the edge to replicate $n-1$. This claim is based on the component-type isomorphism issue where we currently have two identical replicates with no edges. Therefore, we only need to allow a connection to $n-1$ and not to n, and the algorithm will continue generating $\mathcal{G}$. When $n = 1$, an edge will always be allowed because there is no other replicate to compare against.

Since we want to limit connections with this enhancement, we can construct the appropriate vector similar to the rows of the expanded potential adjacency matrix and then include this vector on line 4. This enhancement is implemented efficiently with the `circshift` function [47] and by ensuring that the initial

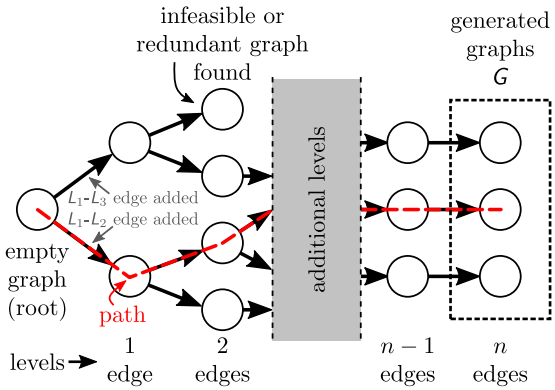
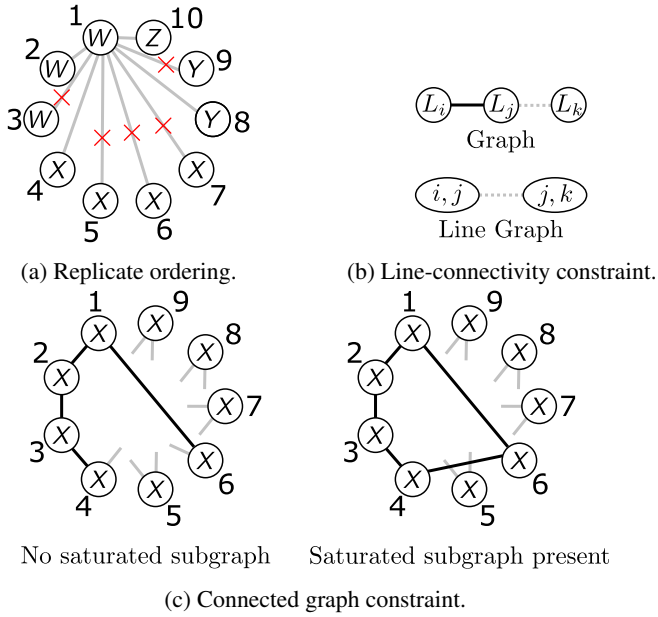


FIGURE 2: Spanning tree representation of Algorithm 1.



(a) Replicate ordering.

(b) Line-connectivity constraint.

(c) Connected graph constraint.

FIGURE 3: Visualizations for some enhancements for enumerating a single catalog.

replicate connections are always allowed. An example is shown in Fig. 3a for the initial iteration and 5/9 edges are not allowed because of this enhancement. Please see Ref. [47] for additional details and examples. The more replicates, the more effective this enhancement will be at reducing the number of generated graphs.

3.3 Loops

A loop is an edge that connects a vertex to itself [40, p. 28]. Using Alg. 1, loops were always required if we wanted to generate graphs without the maximum number of each replicate, even if the feasible graphs should not have any loops [34, 47]. In Ref. [47], a special condition was determined so that loops could be disallowed without compromising $\mathcal{G}$. However, with the change to subcatalogs as discussed in Sec. 2, replicates no longer need to be removed to generate $\mathcal{G}$. Therefore, we can more naturally decide if loops are desired or not. Specifying if loops are allowed can be done for each component type.

In order to not conflict with the reduced potential adjacency matrix $\boldsymbol{A}_r$ that disallows connections between component types, the diagonal elements of the expanded potential adjacency matrix $\boldsymbol{A}$ are based on the user loops specification only. For example, we can now disallow connections $X-X$, but still allow X with loops. This enhancement can reduce the number of graphs generated and should be seen as a compliment to $\boldsymbol{A}_r$ for limiting direct connections.

3.4 Multiedges

A multiedge is formed when two or more edges are incident to the same two vertices [40, p. 28]. In Ref. [34], a multiedge NSC was enforced but only as a FTC, which was extremely inefficient. However, this NSC can and should be implemented as a BTC as is now discussed.

Consider when a component type is required to have unique connections (no multiedges). Due to the sequential nature of Alg. 1, a single edge must be added between two replicates before a second edge is added (creating a multiedge). Therefore, when the first edge is added between two replicates, we can utilize the expanded potential adjacency matrix $\boldsymbol{A}$ to disallow any future connections between those two specific replicates. Since by definition a feasible graph would not have any multiedges for those specific component types, the algorithm with this enhancement will continue generating $\mathcal{G}$ but without the graphs that will eventually be declared as infeasible because of the multiedge NSC.

For the algorithmic implementation, consider two replicates L_i and L_j. If either is required to have no multiedges, then after the initial edge is added between them, we can zero the appropriate entries $\mathsf{A2}_{i,j} = \mathsf{A2}_{j,i} = 0$ of the local expanded potential adjacency matrix after line 11 in Alg. 1. Now, no subsequent iterations will be allowed to add another edge between L_i and L_j. Please see Ref. [47] for additional details and examples.

3.5 Line-connectivity Constraints

The line graph of G is a graph with the edges of G as its vertices, and where two edges of G are adjacent in the line graph if and only if they are incident in G [45, p. 10]. Consider the graph and its corresponding line graph in Fig. 3b with three numbered component types. If L_i is connected to L_j, we can specify if a connection between types L_j and L_k is allowed. This is equivalent to specifying if line type (i, j) can be connected to line type (j,k).

For each line-connectivity constraint, a triple of integers is supplied representing the included component types. Therefore, each triple (i, j, k) is interpreted as: if types L_i and L_j are connected, do not allow connections between types L_j to L_k. These triples help construct the reduced 3-D array (similar to the reduce 2-D array $\boldsymbol{A}_r$) that can be expanded and sliced in a similar manner as $\boldsymbol{A}$ to limit potential connections. Since this NSC requires knowledge of the current edge, it is implemented after line 11 in Alg. 1 on the local expanded potential adjacency matrix $\mathsf{A2}$, similar to the multiedge constraint in Sec. 3.4. Since this limits branches in Alg. 1, it is a BTC. With a suitable number of line-connectivity constraints, conditions such as the following can be enforced: if types L_i and L_j are connected, then type L_j must be connected to type L_k. Please see Ref. [47] for additional details and examples. The engineering applications in Refs. [10, 13, 24, 34] have utilized these constraints to great success.

3.6 Connected Saturated Subgraphs

A subgraph of a graph G is another graph formed from a subset of the vertices and edges of G [40, p. 3]. A saturated subgraph is a subgraph with no empty ports which may contain multiple connected saturated subgraphs (CSSs) [27]. Since a saturated subgraph has no empty ports, no new components can be connected to any of the components in this subgraph during further iterations of the graph generation procedure. This property allows us to include a few additional enhancements.

These enhancements are inspired by the work in Ref. [27] for enumerating molecules where graphs are required to be connected with all atoms present. Therefore, the detection of a saturated subgraph before all atoms have all connections filled indicates the current graph will be infeasible and can be discarded because the resulting graph cannot be a connected graph. Here we allow the specification of more general conditions on saturated subgraphs in $\mathcal{G}$.

First, we can bound the number of CSSs. Requiring at most one CSS is equivalent to requiring a connected graph. These bounds can also be interpreted as the number of connected groupings we desire in $\mathcal{G}$. The upper bound is a BTC while the lower bound is implemented as a FTC. Second, each CSS is composed a specific subcatalog of the original $(\boldsymbol{L}, \boldsymbol{P}, \boldsymbol{R})$. Therefore, the subcatalog constraints described Sec. 2.3 can also be specified for any CSS in $\mathcal{G}$. Both the paired and conflicting component

constraints are now more stringently *path constraints*, where we require a path to exist, or not exist, between the two component types. CSS subcatalog constraints are implemented as a BTC but with the condition that the constraints are only checked when the current graph is a saturated subgraph (so could be considered both a BTC and FTC). Additionally, all constraints are uniformly applied across all CSSs.

To detect if the current graph is a saturated subgraph, we simply check if the replicates can be categorized as having no remaining ports (in the saturated subgraph) or their original port counts (not in the saturated subgraph): $all(\neg \mathsf{V} \,||\, \neg(\mathsf{Vo} - \mathsf{V}))$ where Vo is the original port counts. An example is shown in Fig. 3c. Please see Ref. [47] for additional details and examples. Note that in Ref. [47], this enhancement focused more on the handling of mandatory components (no longer necessary), was able to only specify if a single CCS should be present or not, and the general subcatalog constraints were not included.

4 IMPROVEMENTS ON SET-BASED LABELED GRAPH ISOMORPHISM CHECKING

The graph generation algorithms in this work are not perfect generators for the general component catalog problem, so they may produce graphs that are identical in the context of labeled graph isomorphisms. The issue of isomorphic graphs in the context of engineering applications has been discussed in Refs. [5,6,26,29], but the specific structure of the set of graphs is typically not effectively utilized to create more efficient algorithms.

Definition 1 (Labeled Graph Isomorphism). *Consider two labeled graphs $G_1 = (\boldsymbol{A}_1, \boldsymbol{L}_1)$ and $G_2 = (\boldsymbol{A}_2, \boldsymbol{L}_2)$. G_1 and G_2 are isomorphic if and only if there exists a permutation matrix $\boldsymbol{P}_\pi$ such that 1) $\boldsymbol{A}_1 = \boldsymbol{P}_\pi' \boldsymbol{A}_2 \boldsymbol{P}_\pi$ and 2) $\text{diag}(\boldsymbol{L}_1) = \boldsymbol{P}_\pi' \text{diag}(\boldsymbol{L}_2) \boldsymbol{P}_\pi$ where $\text{diag}(\boldsymbol{L})$ is a diagonal matrix with $\boldsymbol{L}$ as the diagonal elements.*

In this work, we seek the largest subset of graphs $\mathcal{G}$ from the generated set of graphs G where no two graphs in $\mathcal{G}$ are isomorphic based on Def. 1. To determine $\mathcal{G}$, a basic comparison approach would require between n (only one unique graph) and $n^2 + n$ (all graphs unique) pairwise labeled graph isomorphism checks (LGICs) with an algorithm such as `vf2` [50].

However, the number of LGICs can be greatly reduced through the use of graph invariants. Graph invariants are graph properties which are equivalent for isomorphic graphs (i.e., they are necessary conditions) [40, p. 3]. The ones utilized in this work include the number of vertices and edges, determinant, label sequence, labeled degree sequence, labeled loop sequence, labeled connected component sequence, spectrum, and label-shifted spectrum. In Ref. [34], the graph invariants where termed preliminary isomorphism checks and only the number of vertices/edges and label sequences were considered. The labeled sequences (which all have the same number elements) are constructed by sorting the sequence with respect to a canonical $\boldsymbol{D}^L$ (e.g., alphabetical order). For the labeled connected component sequence, all connected components are identified and the labels within each subgraph are sorted for a graph invariant representation. For example, consider two graphs with the following labeled connected component sequences $[A\ B \mid A\ B\ C]$ and $[B\ C\ A \mid B\ A]$ where $\mid$ signifies the start of a new connected component grouping. Utilizing a canonical ordering, these two graphs may be isomorphic because their labeled connected component sequences are identical while a third graph with the sequence $[A\ A\ B \mid B\ C]$ cannot be isomorphic to either of the first two graphs. The effectiveness of graph invariants in reducing the number of LGICs is due to the fact that they are 1) easily computable, 2) only need to be computed once, and 3) a candidate graph's invariants can be compared efficiently to all other graphs through matrix operations.

The most effective graph invariants were found to be the spectrum and novel label-shifted spectrum where the spectrum $\rho(\boldsymbol{A})$ is an ordered sequence of eigenvalues of $\boldsymbol{A}$ [45]. To define the label-shifted spectrum, first consider the following theorem.

Theorem 1. *Let $\boldsymbol{\Lambda}$ be a permutation invariant matrix where $\boldsymbol{\Lambda} = \boldsymbol{P}_\pi' \boldsymbol{\Lambda} \boldsymbol{P}_\pi$ for any permutation matrix $\boldsymbol{P}_\pi$. If $\rho(\boldsymbol{A}_1) = \rho(\boldsymbol{A}_2)$, then $\rho(\boldsymbol{A}_1 + \boldsymbol{\Lambda}) = \rho(\boldsymbol{A}_2 + \boldsymbol{\Lambda})$.*

Proof. Two matrices $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ are cospectral if and only if there exists a permutation matrix $\hat{\boldsymbol{P}}_\pi$ such that $\boldsymbol{A}_1 = \hat{\boldsymbol{P}}_\pi' \boldsymbol{A}_2 \hat{\boldsymbol{P}}_\pi$ [45, p. 164]. For the case of interest:

$$\boldsymbol{A}_1 + \boldsymbol{\Lambda} = \hat{\boldsymbol{P}}_\pi' \boldsymbol{A}_2 \hat{\boldsymbol{P}}_\pi + \boldsymbol{\Lambda} = \hat{\boldsymbol{P}}_\pi' \boldsymbol{A}_2 \hat{\boldsymbol{P}}_\pi + \hat{\boldsymbol{P}}_\pi' \boldsymbol{\Lambda} \hat{\boldsymbol{P}}_\pi \tag{4a}$$

$$= \hat{\boldsymbol{P}}_\pi' (\boldsymbol{A}_2 + \boldsymbol{\Lambda}) \hat{\boldsymbol{P}}_\pi \tag{4b}$$

Therefore a valid permutation matrix is $\hat{\boldsymbol{P}}_\pi$, and the two matrices $\boldsymbol{A}_1 + \boldsymbol{\Lambda}$ and $\boldsymbol{A}_2 + \boldsymbol{\Lambda}$ are cospectral due to the existence of an appropriate permutation matrix. ∎

Since $\rho(\boldsymbol{A})$ does not account for the labels in any way, the idea behind the labeled-shifted spectrum is to introduce perturbations that are graph invariant with respect to the labels. While infinitely many valid $\boldsymbol{\Lambda}$ exist, here we consider the form where the labels are mapped to unique (small) integers which compose the diagonal entries of a diagonal matrix (similar to Definition 1). Due to the errors in numerical computation, a relative error tolerance condition is defined:

$$\frac{||\rho(\boldsymbol{A}_1 + \boldsymbol{\Lambda}) - \rho(\boldsymbol{A}_2 + \boldsymbol{\Lambda})||_\infty}{||\rho(\boldsymbol{A}_1 + \boldsymbol{\Lambda})||_\infty} \leq \epsilon \tag{5}$$

where $\epsilon = 10^{-6}$. Further investigations into the appropriate $\boldsymbol{\Lambda}$ and ϵ should be conducted.

Additionally, in Ref. [34], a "port-type isomorphism filter" was defined as an initial test to determine if any pair of graphs were isomorphic using only the identity permutation matrix $\boldsymbol{P}_\pi$. This type of trivial isomorphism is present in the graph generators, but their occurrence is greatly minimized with the enhancements described in this article. This is efficiently computed by finding the unique rows in a matrix with rows representing the flattened adjacency matrices of each graph with the same $\boldsymbol{D}^L$.

ALGORITHM 3: Determine set of unique labeled graphs.

Input : G – set of labeled graphs
Output: $\mathcal{G}$ – set of unique labeled graphs

```
G ← Check for trivial isomorphisms in G                    // port-type LGIC
Compute graph invariants for all G
Iunique ← 1                                                // first graph is always unique
for k ← 2 to length(G) do
    Icheck ← Determine all G(Iunique) that have same graph
      invariants to G(k)
    IsIso ← false                                          // initialize graph as unique
    for i ← Icheck do                                      // check for graph isomorphism
        IsIso ← Check if G(k) and G(Icheck(i)) are isomorphic
        Terminate loop if IsIso is true, otherwise continue
    end
    if IsIso is false then
        Iunique(end + 1) ← k                               // add unique graph
    end
end
𝒢 ← G(Iunique)                                             // extract unique graphs
```

The general algorithm for determining the set of nonisomorphic graphs from a given set is presented in Alg. 3 and is quite similar to the one shown in Ref. [34] without bins. This algorithm and `matlab` implementation in Ref. [43] could be used with any set of labeled graphs, including those created with alternative graph generation methods.

5 GRAPH ENUMERATION EXAMPLES

In this section, a variety of examples are presented using the component catalog representation and enumeration algorithms described in the previous sections. These examples represent both new and existing graph enumeration problems from both mathematics and engineering with the intention to demonstrate the breadth of graph enumeration problems that can be posed and demonstrate the ways in which this approach can be considered as domain-independent architecture decision support [1].

The `matlab` tool developed using the methods described in this article is available at Ref. [43]. The computer architecture used in obtaining the results for all case studies was a desktop workstation with an i7-6800K CPU at 3.8 GHz, 32 GB DDR4 3200 MHz RAM, `matlab` R2020a update 2, `python` 3.8.3, `python-igraph` 0.8.2, and `windows` 10 build 18363.836.

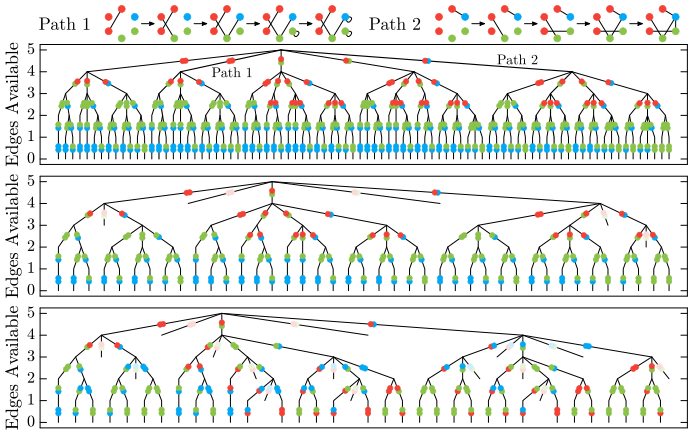
5.1 Case Studies from the Original Paper

Three case studies were presented in Ref. [34].

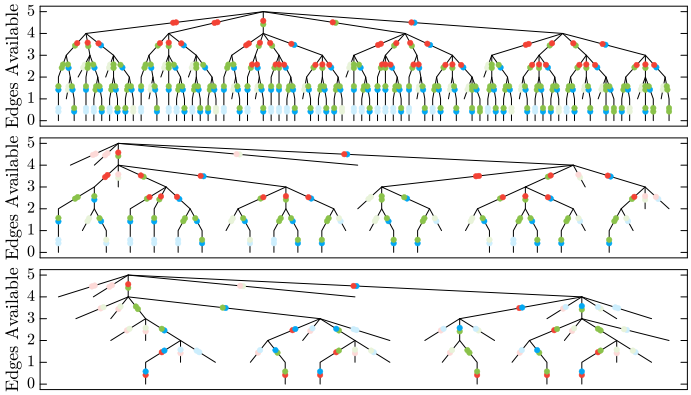
5.1.1 Case Study 1 The following component catalog was the first case study in Ref. [34]:

$$\boldsymbol{L} = [R\ G\ B], \quad \underline{\boldsymbol{P}} = \bar{\boldsymbol{P}} = [1\ 2\ 3], \quad \underline{\boldsymbol{R}} = \bar{\boldsymbol{R}} = [3\ 2\ 1] \qquad (6)$$

The results for two different problem variations with Catalog (6) are shown in Table 1. In both variations, both the number of candidate graphs generated and computational cost were decreased.



(a) Catalog (6) spanning tree using Alg. 1 (top), with enhancements except touched vertex promotion (middle), and all enhancements (bottom).



(b) Catalog (6) with NSCs spanning tree using Alg. 1 (top), with enhancements except touched vertex promotion (middle), and all enhancements (bottom).

FIGURE 4: Case Study 1 spanning trees with different implementations.

This problem is small enough that the spanning tree representation portrayed in Fig. 2 can be visualized for both problem variations and different algorithm implementations. In Fig. 4a, the spanning trees using Alg. 1 and versions utilizing the enhancements with and without touched vertex promotion are shown. The paired nodes on the branches indicate which component types were connected with this additional edge. Branches that have faded paired nodes (and no children) indicate that either the branch was determined to be isomorphic to another branch or a NSC was violated. The original algorithm has no terminated branches, while a few occur early with the enhancements. Even though only a few branches were removed, because of the growth properties of the algorithm, the number of generated graphs is significantly smaller.

Figure 4b shows the spanning tree for one subcatalog with the additional NSCs described in Table 1. Since loops were not

TABLE 1: Comparison between original and current methods for the Case Studies from the Original Paper.

Additional NSCs		NST	CG	NIFG	UFG	LGIC	t (s)
Catalog (6)	original	244	86	77	16	153	0.050
	current	95	33	33	16	17	0.013
Catalog (6), connected, no multiedges, no loops, $\underline{\boldsymbol{R}} = [0\ 0\ 1]$	original	244	86	23	5	23	0.028
	current	59	11	11	5	6	0.016
Catalog (7)	original	3441	1119	767	274	12948	2.481
	current	1804	629	338	274	64	0.084
Catalog (7), connected, no multiedges, no loops, $\underline{\boldsymbol{R}} = [1\ 0\ 0\ 0\ 0]$	original	3441	1119	767	140	2007	0.466
	current	1632	393	190	140	50	0.082
Catalog (7), connected, no multiedges, no loops	original	3441	1119	31	12	100	0.156
	current	387	54	22	12	10	0.016
Catalog (7), connected, no multiedges, no loops, $\underline{\boldsymbol{R}} = [1\ 2\ 1\ 1\ 1]$	original	3441	1119	34	14	102	0.153
	current	493	64	25	14	11	0.022
Catalog (8)	original	1761015019	158154694	1943862	12480	28471024	17903.200
	current	5230799	210637	40432	12480	27952	18.433

NST: nodes in spanning tree, CG: candidate graphs from enumeration algorithm, NIFG: non-trivially isomorphic and feasible graphs, UFG: unique feasible graphs, LGIC: labeled graph isomorphism checks

allowed, Alg. 1 employed the appropriate $\boldsymbol{A}_r$, and a number of branches were eliminated. The connected saturated subgraph enhancement greatly reduced the number of generated graphs. Observing the bottom spanning tree, nearly all potential branches were removed. All three spanning trees displayed the property that the number of intermediary graphs grows at first but then decreases due to the NSCs. For example, the middle tree has $2 \to 6 \to 14 \to 17 \to 8$ vertices with children (feasible partial graphs) at each level.

5.1.2 Case Study 2

The following component catalog was the second case study in Ref. [34]:

$$\boldsymbol{L} = [P\ R\ G\ B\ O] \tag{7a}$$

$$\underline{\boldsymbol{P}} = \bar{\boldsymbol{P}} = [1\ 1\ 2\ 3\ 4], \quad \underline{\boldsymbol{R}} = \bar{\boldsymbol{R}} = [1\ 2\ 2\ 1\ 1] \tag{7b}$$

The results for the four different problem variations are shown in Table 1. Again, the number of candidate graphs from the enumeration algorithm, feasible and not trivially isomorphic graphs, and isomorphism checks were greatly decreased while still producing the same desired $\mathcal{G}$. These reductions translated into noticeable computational cost reductions (between 6 and 30× overall speedups).

Impressively, every isomorphism check had a 100% hit rate, i.e., when line 8 in Alg. 3 was reached, the two graphs were determined to be isomorphic. This is substantially better than the original methods and represents a considerable computational cost savings. This is a testament to the effectiveness of the improvements discussed in Sec. 4.

5.1.3 Suspension Case Study

The primary engineering-focused case study in Ref. [34] was of a quarter car vehicle suspension. Different graphs representing the physical behavior of different suspension architecture concepts were expressed using the following catalog:

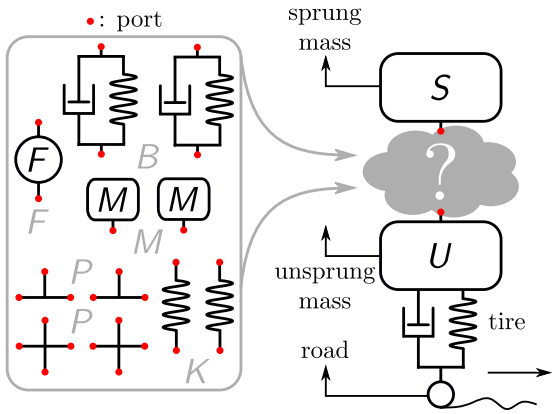
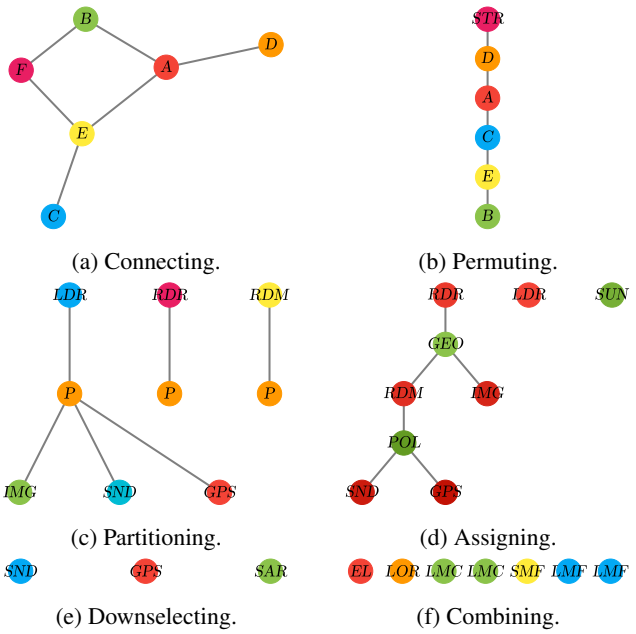


FIGURE 5: Suspension Case Study catalog visualization.

$$\boldsymbol{L} = [S\ U\ M\ K\ B\ F\ P\ P], \quad \underline{\boldsymbol{P}} = \bar{\boldsymbol{P}} = [1\ 1\ 1\ 2\ 2\ 2\ 3\ 4] \tag{8a}$$

$$\underline{\boldsymbol{R}} = [1\ 1\ 0\ 0\ 0\ 1\ 0\ 0], \quad \bar{\boldsymbol{R}} = [1\ 1\ 2\ 2\ 2\ 1\ 2\ 2] \tag{8b}$$

where no multiedges or loops are allowed, a connected graph is required, and several direct connection constraints and line-connectivity constraints are specified. The additional custom NSCs included conditions that a path between U and S must exist with at least one (K, B, F), no cycles that contain only a single P, and only specific orderings of the series connections are allowed because permuting series connections results in an equivalent model (e.g., $K-B$ and $B-K$ in series are physically equivalent). The catalog is visualized in Fig. 5. For more details,



(a) Connecting. (b) Permuting.

(c) Partitioning. (d) Assigning.

(e) Downselecting. (f) Combining.

FIGURE 6: Example graphs representing the different patterns.

please see Refs. [34, 43].

With all the enhancements, the number of candidate graphs was reduced 751×, the number of feasible graphs reduced 48×, and the number of isomorphism checks reduced 1019×. This translated into a computational cost decrease of 971×. Again, a 100% isomorphic checking hit rate was observed. This case study demonstrates that the generation of a complex $\mathcal{G}$ can be done efficiently, and the still too common notion that enumeration can only be performed with naïve methods should be reconsidered.

5.2 Patterns in System Architecture Decisions

In Ref. [1], six canonical classes of architectural decisions, termed patterns, were presented, namely the connecting, permuting, partitioning, downselecting, combining, and assigning patterns. In this section, we will show that these patterns can be represented using the problem definitions in this work (similar to the mappings between the patterns in Ref. [1]). However, it is important to note that the considered implementation might not be the most efficient method for generating architecture candidates for a particular pattern. However, the work presented here and tools developed from it can play a role as the domain-independent architecture decision support tool desired in Ref. [1].

5.2.1 Connecting Pattern

This pattern seeks all edge lists for a given set of vertices. While no specific problem was given in Ref. [1], a generic connecting pattern problem is:

$$\boldsymbol{L} = [A \overset{n}{\cdots} Z], \quad \underline{\boldsymbol{P}} = [1 \overset{n}{\cdots} 1], \quad \bar{\boldsymbol{P}} = [n \overset{n}{\cdots} n] \tag{9a}$$

$$\underline{\boldsymbol{R}} = [1 \overset{n}{\cdots} 1], \quad \bar{\boldsymbol{R}} = [n \overset{n}{\cdots} n] \tag{9b}$$

where no multiedges or loops are allowed. With no other NSCs, this is equivalent to A006125, number of graphs on n labeled nodes [39]. The number of unique graphs growths extremely fast with $|\mathcal{G}| = 2^{(n^2-n)/2}$; there are already 32768 graphs when $n = 6$ (73.0 s) and essentially captures all other patterns for large enough n since we are generating all possible adjacency matrices. As stated in Ref. [34], "with this generality comes an enormous space, potentially too large to be useful for certain graph problems". The primary motivator for the original work on Alg. 1 in Ref. [34] was to move away from adjacency matrix permutations towards a more useful construction of the desirable graphs. The PM-based approach was shown to have much more desirable properties for the set of graphs defined by a component catalog and was compared to index stack blocks [29], an adjacency matrix permutation method. Most of the enhancements in this article leverage the specified catalog structure in some way.

In Ref. [1], connecting subpatterns termed architecture styles were defined to add more graph structure. Simpler architecture styles such as the trees, stars, and rings can be more easily defined using a component catalog and some NSCs, while more complex styles, such as a mesh, may be more challenging to effectively represent.

5.2.2 Permuting Pattern

This pattern seeks to arrange a set of elements in different orderings [1]. The example from Ref. [1] simply contains five generic elements:

$$\boldsymbol{L} = [\mathit{STR}\ A\ B\ C\ D\ E],\ \underline{\boldsymbol{P}} = [1\ 1 \overset{5}{\cdots} 1],\ \bar{\boldsymbol{P}} = [1\ 2 \overset{5}{\cdots} 2] \tag{10a}$$

$$\underline{\boldsymbol{R}} = \bar{\boldsymbol{R}} = [1\ 1 \overset{5}{\cdots} 1] \tag{10b}$$

where no multiedges or loops are allowed. The label STR indicates the starting point of the permutation, and the component type with one port is the endpoint. With no other NSCs, this is equivalent to A000142 (number of permutations of n vertices) [39]. Since $n = 5$ in Catalog (10), we have 120 unique permutations (0.03 s). Here we note that the graph generator only generates unique graphs so isomorphism checking is not strictly required.

Many potential NSCs can be included. Direct connection constraints could ensure that two types are not adjacent to one another. Line-connectivity constraints could constrain relative position of nodes with respect to the greedy/incremental architecture styles [1]. Custom subcatalog NSCs can be defined if the component types have different costs associated with their absolute and relative positions.

5.2.3 Partitioning Pattern

This pattern occurs when there is a set of entities that need to be grouped into nonoverlapping subsets [1]. The example from Ref. [1] is concerned with placing

instruments on different spacecraft (represented by partitions P). All non-P types in the catalog are different instruments, and this partitioning problem is represented by:

$$\boldsymbol{L} = [P\ LDR\ RDR\ RDM\ IMG\ SND\ GPS] \tag{11a}$$

$$\underline{\boldsymbol{P}} = [1\ 1\ \overset{6}{\cdots}\ 1], \quad \bar{\boldsymbol{P}} = [6\ 1\ \overset{6}{\cdots}\ 1] \tag{11b}$$

$$\underline{\boldsymbol{R}} = [1\ 1\ \overset{6}{\cdots}\ 1], \quad \bar{\boldsymbol{R}} = [6\ 1\ \overset{6}{\cdots}\ 1] \tag{11c}$$

where multiedges and loops are not allowed, no $P-P$ connections allowed, all non-P types can only be connected to P, and the subcatalog filters for bipartite graphs from Sec. 2.3 can be used. With no other NSCs, this is equivalent to A000110, number of ways to partition a set of n labeled elements [39]. Since $n = 6$ in Catalog (11), we have 203 unique partitions (0.07 s).

Again, many potential engineering NSCs can be readily added such as the minimum/maximum number of partitions through $\underline{\boldsymbol{R}}_1$ and $\bar{\boldsymbol{R}}_1$, respectively. Similarly, the minimum/maximum number of elements in a partition can be limited through $\underline{\boldsymbol{P}}_1$ and $\bar{\boldsymbol{P}}_1$, respectively. Using line-connectivity constraints, we can decide if two types can be in the same partition. A custom NSC could be defined to limit a maximum cost for each partition. Such additional NSCs can have the benefit of improving the usefulness of the generated graphs and reducing computational cost. For example, consider the number of partitions for $n = 10$ with the modifications limiting $\mathcal{G}$ to only contain $2-5$ partitions with $2-3$ elements. Then there are 7245 unique partitions found in 0.75 s vs. 115975 partitions found in 10.50 s without the modifications.

5.2.4 Downselecting Pattern

This pattern seeks a subset of entities among a set of options. The example from Ref. [1] is concerned with selecting which instruments should be selected for an Earth observing system, and this downselecting problem is represented by:

$$\boldsymbol{L} = [LDR\ RDR\ RDM\ IMG\ SND\ GPS\ SAR\ SPM] \tag{12a}$$

$$\underline{\boldsymbol{P}} = \bar{\boldsymbol{P}} = [2\ \overset{8}{\cdots}\ 2], \quad \underline{\boldsymbol{R}} = [0\ \overset{8}{\cdots}\ 0], \quad \bar{\boldsymbol{R}} = [1\ \overset{8}{\cdots}\ 1] \tag{12b}$$

where only loops are allowed (limited using $\boldsymbol{A}_r$). With no other NSCs, this is equivalent to A000079, number of binary vectors of length n [39]. Since $n = 8$ in Catalog (11), we have 256 unique graphs (0.38 s). We note that this computational cost includes a fair amount of overhead because we are directly computing a graph representations of these binary vectors and tailored binary enumeration methods could be more efficient (or simply not generating the graphs but only the set of subcatalogs which only took 0.01 s).

Again, additional NSCs can be readily added such as linear penalty constraints to limit the maximum system cost or number or replicates and linear satisfaction constraints to ensure that a downselection meets all system requirements. For example, consider the cost linear penalty constraint captured by $\boldsymbol{\rho} = [4,5,5,3,3,6,4,4]$ and $\alpha = 10$, then there are only 38 graphs (0.07 s). Additionally, $\bar{\boldsymbol{R}}$ can be modified to define the number of maximum replicates to include if multiple copies of the same component type can be included. NSCs in this pattern are really only applicable to the subcatalog since each valid subcatalog has exactly one unique graph. Again, alternative tailored tools such as ones for the $0-1$ knapsack problem could be more effective for this type of pattern [1].

5.2.5 Combining Pattern

This pattern seeks a combination of exactly one option for each decision. The example from Ref. [1] is based on Ref. [2] for different decisions for the Apollo program:

$$\boldsymbol{L} = [EOR\ EL\ LOR\ MA\ MD\ CMC\ LMC\ SMF\ LMF] \tag{13a}$$

$$\underline{\boldsymbol{P}} = \bar{\boldsymbol{P}} = [2\ \overset{9}{\cdots}\ 2],\ \underline{\boldsymbol{R}} = [0\ \overset{9}{\cdots}\ 0],\ \bar{\boldsymbol{R}} = [1\ 1\ 1\ 1\ 1\ 1\ 3\ 1\ 2] \tag{13b}$$

where only loops are allowed (limited using $\boldsymbol{A}_r$) and $\bar{\boldsymbol{R}}$ represents the number of alternatives minus one. Therefore, the selected alternative for each type is simply related to the number of replicates in the graph. Here, we have 1536 unique combinations (0.04 s), equivalently defined as the Cartesian product of the decision sets. Including the Apollo logical constraints from Ref. [2] as custom subcatalog constraints reduces number of valid mission plans to 108 (0.04 s).

All types of subcatalog constraints could be included to restrict $\mathcal{G}$. However, this representation is certainly a bit cumbersome, but directly allowing for set definitions in the catalog, as discussed in Sec. 2.1, would facilitate better the inclusion of NSCs with the combining pattern.

5.2.6 Assigning Pattern

This patterns seeks to assign one set of entities to another set. The example from Ref. [1] seeks to assign instruments to orbits for an Earth observing system:

$$\boldsymbol{L} = [\overbrace{LDR\ RDR\ RDM\ IMG\ SND}^{\text{instruments}}\ \overbrace{GEO\ SUN\ POL}^{\text{orbits}}] \tag{14a}$$

$$\underline{\boldsymbol{P}} = [0\ \overset{5}{\cdots}\ 0\ 0\ \overset{3}{\cdots}\ 0],\ \bar{\boldsymbol{P}} = [3\ \overset{5}{\cdots}\ 3\ 5\ \overset{3}{\cdots}\ 5] \tag{14b}$$

$$\underline{\boldsymbol{R}} = \bar{\boldsymbol{R}} = [1\ \overset{5}{\cdots}\ 1\ 1\ \overset{3}{\cdots}\ 1] \tag{14c}$$

where no multiedges or loops are allowed, instrument-to-instrument connections, no orbit-to-orbit connections, and the subcatalog filters for bipartite graphs from Sec. 2.3 can be used. There are many feasible subcatalogs (51711) for the 262144 graphs (700.63 s) for Catalog (14). The normal use of parallel computing with respect to this many subcatalogs was found to be inefficient.

Again, once some reasonable NSCs are added, the number of graphs decreases significantly, and the methods described in this work become much more attractive. For example, consider the case with precisely 1 replicate of each instrument, and a given orbit can have a maximum of 3 instruments. Now the time needed to generate the 510 unique feasible graphs is only 0.06 s.

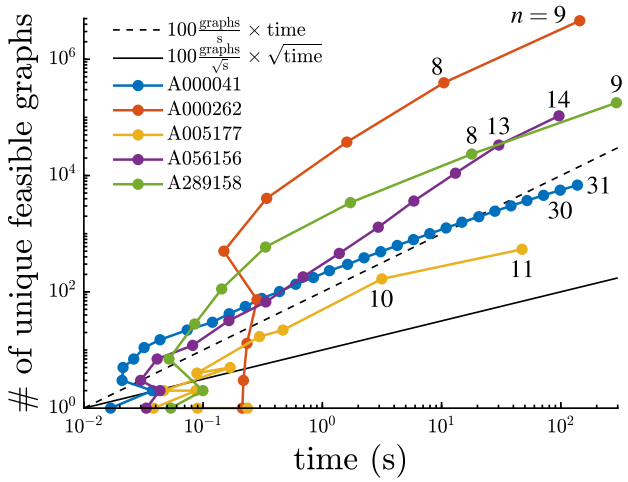


FIGURE 7: Number of unique feasible graphs for the OEIS examples vs. computational time (multiple values of n shown).

5.3 Additional Graph Enumeration Examples

The On-Line Encyclopedia of Integer Sequences (OEIS) is a database containing thousands of integer sequences, many corresponding to different graph enumeration problems. Here, five different sequences are replicated using the problem definitions in this work and can serve as some validation for the presented algorithms as the number of graphs is the same between OEIS and the results of this work. Additionally, these examples provide significant insights into how to pose new problems using the component catalog definition. The number of UFGs vs. computational time is shown in Fig. 7. Note the variation in the number of UFG graphs generated per second but generally similar growth rates. Many more OEIS examples can be found in the code repository [43].

5.3.1 A000041 This sequence represents the number of partitions of a number n [39]. This sequence can be generated by counting all graphs with the following catalog:

$$\boldsymbol{L} = [O], \quad \underline{\boldsymbol{P}} = \bar{\boldsymbol{P}} = [2], \quad \underline{\boldsymbol{R}} = \bar{\boldsymbol{R}} = [n] \tag{15}$$

where multiedges and loops are allowed. The sequence values were validated up to $n = 31$ (139 s).

5.3.2 A000262 This sequence represents all labeled rooted skinny-tree forests on n vertices [39]. This class of graphs was used in Ref. [25]. All graphs in this sequence can be generated with the following catalog:

$$\boldsymbol{L} = [ROOT\ A \overset{n}{\cdot\cdot} Z], \quad \underline{\boldsymbol{P}} = [1\ 1 \overset{n}{\cdot\cdot} 1] \tag{16a}$$

$$\bar{\boldsymbol{P}} = [n\ 2 \overset{n}{\cdot\cdot} 2], \quad \underline{\boldsymbol{R}} = \bar{\boldsymbol{R}} = [1 \overset{n}{\cdot\cdot} 1] \tag{16b}$$

where no multiedges or loops are allowed, a connected graph is required, and the number of ports must be equal to $2n$ (tree graph condition). The sequence values were validated up to $n = 9$ (144 s).

5.3.3 A005177 This sequence represents the number of connected regular graphs with n nodes [39]. All graphs in this sequence can be generated with the following catalog:

$$\boldsymbol{L} = [A], \quad \underline{\boldsymbol{P}} = [0], \quad \bar{\boldsymbol{P}} = [n-1], \quad \underline{\boldsymbol{R}} = \bar{\boldsymbol{R}} = [n] \tag{17}$$

where no multiedges or loops are allowed, a connected graph is required, and a custom subcatalog function is used to filter out any subcatalogs that do not have the same number of ports for every replicate (regular graph condition). The sequence values were validated up to $n = 11$ (47 s).

5.3.4 A056156 This sequence represents the number of connected bipartite graphs with n edges, no isolated vertices, and a distinguished bipartite block, up to isomorphism [39]. All graphs in this sequence can be generated with the following catalog:

$$\boldsymbol{L} = [A\ B], \ \underline{\boldsymbol{P}} = [1\ 1], \ \bar{\boldsymbol{P}} = [n\ n], \ \underline{\boldsymbol{R}} = [1\ 1], \ \bar{\boldsymbol{R}} = [n\ n] \tag{18}$$

where no multiedges or loops are allowed, a connected graph is required, only $A-B$ connections allowed using $\boldsymbol{A}_r$, the number of ports must be equal to $2n$ (double the number of edges), and the subcatalog filters for bipartite graphs from Sec. 2.3 can be used. The sequence values were validated up to $n = 14$ (97 s).

5.3.5 A289158 This sequence represents the number of connected multigraphs with n nodes of degree at most 4 and with at most double edges [39]. In chemistry, these graphs represent molecules (excluding stereoisomers) without triple bonds, given n carbon atoms. All graphs in this sequence can be generated with the following catalog:

$$\boldsymbol{L} = [C], \quad \underline{\boldsymbol{P}} = [0], \quad \bar{\boldsymbol{P}} = [4], \quad \underline{\boldsymbol{R}} = \bar{\boldsymbol{R}} = [n] \tag{19}$$

where no loops are allowed, a connected graph is required, and the maximum number of multiedges is limited to 2 using the methods in Sec. 3.4. The sequence values were validated up to $n = 9$ (293 s).

5.4 Physics-based Engineering Design Case Studies

It is important to note that while the techniques described in this article are quite technical, the tool developed using these techniques has had an impact on several physics-based engineering design case studies that leveraged the tool in Ref. [43].

In Ref. [24], graphs from the vehicle suspension case study in Sec. 5.1.3 were further constrained with additional NSCs, and for every candidate suspension graph, a combined plant and control design problem was solved using a dynamic model automatically generated from the suspension graph representation. In Ref. [10], passive electric circuits were generated, and nonlinear fitting problems were solved to match desired frequency responses and realize low-pass filters. In Ref. [25], different single-split fluid-based thermal management architectures where generated and optimized. While A000262 [39] captured the desired counts of all labeled rooted skinny-tree forests, the explicit

enumeration of the graphs was required. Finally, in Ref. [13], a customized graph enumeration procedure utilizing Ref. [43] was used to generate candidate aircraft air cycle machines for thermal management. Modelica models were then automatically constructed to assess the performance of the given graph. Many of these studies would not have been possible without enhancements to the original Alg. 1.

6 CONCLUSION

This article built upon the perfect matching-inspired, brute-force algorithm presented in Ref. [34] (and implemented in Ref. [43]) for generating all graphs in a specified graph structure space that is suitable for representing many graph-based engineering design challenges. First, a new component catalog definition was proposed that naturally expanded the potential graphs that could be represented. Next, the use of subcatalogs in the generation process was discussed, including the handling of a variety of subcatalog constraints.

Then several enhancements for the enumeration of a specific subcatalog were presented. Some of the enhancements reduced the number of isomorphic graphs generated, such as the BFS implementation, sorted catalogs, and replicate ordering while others formalized certain specific NSCs in the graph generation algorithm. Finally, because the graph generators are not perfect, some of the generated graphs may not be unique. However, the effective use of graph invariants, including the label-shifted spectrum, significantly reduced the number of expensive labeled graph isomorphism checks to only a few pairs which had an extremely high probability of being isomorphic. All of these enhancements worked together to produce an approach that is more parallelized and requires fewer basic and expensive operations.

A variety graph enumeration examples were presented, including examples from the original paper [34], patterns in system architecture decisions [1], OEIS [39], and previous physics-based engineering design problems [10, 13, 24, 25]. In the vehicle suspension case study, an overall computational speedup of 971× was observed compared to the original method. For the patterns, the constrained versions of the patterns were shown to be efficiently generated. However, tailored graph generators could always be more effective than the generalized approach considered here. Additionally, the OEIS sequences provided additional validation and insights into how to pose diverse graph enumeration problems with this approach. Overall, the examples demonstrate the ways this approach can be considered as a domain-independent architecture decision support tool.

Future work items include developing more standardized NSCs to make it easier to define the desired problem, determining better heuristics for the catalog representation and sorting labels, complete inclusion of directed graphs and structured components [34], and investigating randomized generation methods that can produce a suitable sampling of $\mathcal{G}$ for use in population-based optimization or designer-in-the-loop graph exploration for a large $\mathcal{G}$. While the combinatorial growth of enumeration-based methods will always be a persisting issue, the advancements in graph-based tools make the previously unthinkable achievable.

REFERENCES

[1] Selva, D., Cameron, B., and Crawley, E., 2017. "Patterns in system architecture decisions". *Syst. Eng.,* ***19***(6), Nov., pp. 477–497. doi: 10.1002/sys.21370

[2] Simmons, W. L., 2008. "A framework for decision support in systems architecting". PhD thesis, Massachusetts Institute of Technology.

[3] Snavely, G. L., and Papalambros, P. Y., 1993. "Abstraction as a configuration design methodology". In Advances in Design Automation, Vol. 65, pp. 297–305.

[4] Münzer, C., Helms, B., and Shea, K., 2013. "Automatically transforming object-oriented graph-based representations into boolean satisfiability problems for computational design synthesis". *J. Mech. Des.,* ***135***(10), July, p. 101001. doi: 10.1115/1.4024850

[5] Bayrak, A. E., Ren, Y., and Papalambros, P. Y., 2016. "Topology generation for hybrid electric vehicle architecture design". *J. Mech. Des.,* ***138***(8), June, p. 081401. doi: 10.1115/1.4033656

[6] Silvas, E., Hofman, T., Serebrenik, A., and Steinbuch, M., 2015. "Functional and cost-based automatic generator for hybrid vehicles topologies". *IEEE/ASME T. Mech.,* ***20***(4), Aug., pp. 1561–1572. doi: 10.1109/TMECH.2015.2405473

[7] Silvas, E., Hofman, T., Murgovski, N., Etman, P., and Steinbuch, M., 2017. "Review of optimization strategies for system-level design in hybrid electric vehicles". *IEEE Trans. Veh. Technol.,* ***66***(1), Jan., pp. 57–70. doi: 10.1109/TVT.2016.2547897

[8] Docimo, D. J., Kang, Z., James, K. A., and Alleyne, A. G., 2020. "A novel framework for simultaneous topology and sizing optimization of complex, multi-domain systems-of-systems". *J. Mech. Des.,* ***142***(9), Mar. doi: 10.1115/1.4046066

[9] Foster, R. M., 1932. "Geometrical circuits of electrical networks". *Trans. Am. Inst. Electr. Eng.,* ***51***(2), June, pp. 309–317. doi: 10.1109/T-AIEE.1932.5056068

[10] Herber, D. R., 2017. "Advances in combined architecture, plant, and control design". Ph.D. Dissertation, University of Illinois at Urbana-Champaign, Urbana, IL, USA, Dec.

[11] Zeidner, L. E., Reeve, H. M., Khire, R., and Becz, S., 2010. "Architectural enumeration and evaluation for identification of low-complexity systems". In Aviation Technology, Integration, and Operations (ATIO) Conference, no. AIAA 2010-9264. doi: 10.2514/6.2010-9264

[12] Zeidner, L. E., St. Rock, B. E., Desai, N. A., Reeve, H. M., and Strauss, M. P., 2010. "Application of a technology screening methodology for rotorcraft alternative power systems". In Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition, no. AIAA 2010-1505. doi: 10.2514/6.2010-1505

[13] Herber, D. R., Allison, J. T., Buettner, R., Abolmoali, P., and Patnaik, S. S., 2020. "Architecture generation and performance evaluation of aircraft thermal management systems through graph-based techniques". In Science and Technology Forum and Exposition, no. AIAA 2020-0159. doi: 10.2514/6.2020-0159

[14] Pennestrì, E., and Valentini, P. P., 2015. "Kinematics and enumeration of combined harmonic drive gearing". *J. Mech. Des.,* ***137***(12), Oct., p. 122303. doi: 10.1115/1.4031590

[15] Königseder, C., and Shea, K., 2015. "Comparing strategies for topologic and parametric rule application in automated computational design synthesis". *J. Mech. Des.,* ***138***(1), Nov., p. 011102. doi: 10.1115/1.4031714

[16] del Castillo, J. M., 2002. "Enumeration of 1-DOF planetary gear train graphs based on functional constraints". *J. Mech. Des.,* ***124***(4), Nov., pp. 723–732. doi: 10.1115/1.1514663

[17] Ruddigkeit, L., van Deursen, R., Blum, L. C., and Reymond, J.-L., 2012. "Enumeration of 166 billion organic small molecules in the chemical universe database GDB-17". *J. Chem. Inf. Model.,* ***52***(11), Nov., pp. 2864–2875. doi: 10.1021/ci300415d

[18] Carhart, R. E., Smith, D. H., Brown, H., and Djerassi, C., 1975. "Applications of artificial intelligence for chemical inference. XVII. approach to computer-assisted elucidation of molecular structure". *J. Am. Chem. Soc.,* ***97***(20), Oct., pp. 5755–5762. doi: 10.1021/ja00853a021

[19] Faulon, J.-L., 1998. "Isomorphism, automorphism partitioning, and canonical labeling can be solved in polynomial-time for molecular graphs". *J. Chem. Inf. Comput. Sci.,* ***38***(3), Mar., pp. 432–444. doi: 10.1021/ci9702914

[20] Ma, W., Trusina, A., El-Samad, H., Lim, W. A., and Tang, C., 2009. "Defining network topologies that can achieve biochemical adaptation". *Cell,* ***138***(4), Aug., pp. 760–773. doi: 10.1016/j.cell.2009.06.013

[21] Wu, Z., Campbell, M. I., and Fernández, B. R., 2008. "Bond graph based automated modeling for computer-aided design of dynamic systems". *J. Mech. Des.,* ***130***(4), Mar., p. 041102. doi: 10.1115/1.2885180

[22] Schmidt, L. C., Shetty, H., and Chase, S. C., 2000. "A graph grammar approach for structure synthesis of mechanisms". *J. Mech. Des.,* ***122***(4), Dec., pp. 371–376. doi: 10.1115/1.1315299

[23] Maxwell III, J. T., de Kleer, J., and Klenk, M., 2019. "Insane design of lumped element models". In International Workshop on Qualitative Reasoning.

[24] Herber, D. R., and Allison, J. T., 2019. "A problem class with combined architecture, plant, and control design applied to vehicle suspensions". *J. Mech. Des.,* ***141***(10), May. doi: 10.1115/1.4043312

[25] Peddada, S. R. T., Herber, D. R., Pangborn, H. C., Alleyne, A. G., and Allison, J. T., 2019. "Optimal flow control and single split architecture exploration for fluid-based thermal management". *J. Mech. Des.,* ***141***(8), Apr. doi: 10.1115/1.4043203

[26] Hartmann, C., Chenouard, R., Mermoz, E., and Bernard, A., 2018. "A framework for automatic architectural synthesis in conceptual design phase". *J. Eng. Des.,* ***29***(11), Oct., pp. 665–689. doi: 10.1080/09544828.2018.1532494

[27] Faulon, J.-L., Churchwell, C. J., and Visco, D. P., 2003. "The signature molecular descriptor. 2. Enumerating molecules from their extended valence sequences". *J. Chem. Inf. Comput. Sci.,* ***43***(3), May, pp. 721–734. doi: 10.1021/ci020346o

[28] Mittal, S., and Frayman, F., 1989. "Towards a generic model of configuration tasks". In International Joint Conference on Artificial Intelligence, pp. 1395–1401.

[29] Wyatt, D. F., Wynn, D. C., and Clarkson, P. J., 2014. "A scheme for numerical representation of graph structures in engineering design". *J. Mech. Des.,* ***136***(1), Jan., p. 011010. doi: 10.1115/1.4025961

[30] Schmidt, L. C., and Cagan, J., 1997. "GGREADA: a graph grammar-based machine design algorithm". *Res. Eng. Des.,* ***9***(4), Dec., pp. 195–213.

[31] Chakrabarti, A., Shea, K., Stone, R., Cagan, J., Campbell, M., Hernandez, N. V., and Wood, K. L., 2011. "Computer-based design synthesis research: an overview". *J. Comput. Inf. Sci. Eng.,* ***11***(2), June, p. 021003. doi: 10.1115/1.3593409

[32] Campbell, M., 2009. A graph grammar methodology for generative systems. Tech. rep., University of Texas at Austin.

[33] Behbahani, S., and de Silva, C. W., 2013. "Automated identification of a mechatronic system model using genetic programming and bond graphs". *J. Dyn. Syst. Meas. Contr.,* ***135***(5), May, p. 051007. doi: 10.1115/1.4024171

[34] Herber, D. R., Guo, T., and Allison, J. T., 2017. "Enumeration of architectures with perfect matchings". *J. Mech. Des.,* ***139***(5), Apr., p. 051403. doi: 10.1115/1.4036132

[35] Chapman, W. L., Rozenblit, J., and Bahill, A. T., 2001. "System design is an NP-complete problem". *Syst. Eng.,* ***4***(3), pp. 222–229. doi: 10.1002/sys.1018

[36] Wyatt, D. F., Wynn, D. C., Jarrett, J. P., and Clarkson, P. J., 2012. "Supporting product architecture design using computational design synthesis with network structure constraints". *Res. Eng. Des.,* ***23***(1), pp. 17–52. doi: 10.1007/s00163-011-0112-y

[37] Bryant, C. R., McAdams, D. A., Stone, R. B., Kurtoglu, T., and Campbell, M. I., 2005. "A computational technique for concept generation". In International Design Engineering Technical Conferences, no. DETC2005-85323. doi: 10.1115/DETC2005-85323

[38] Rispoli, F. J., 2007. *Applications of Discrete Mathematics*, updated ed. McGraw-Hill, ch. Applications of subgraph enumeration, pp. 241–262.

[39] The on-line encyclopedia of integer sequences (A000041, A000079, A000110, A000142, A000262, A001147, A005177, A006125, A056156, A289158). Online. url: https://oeis.org

[40] Diestel, R., 2017. *Graph Theory*, 5th ed. Springer. doi: 10.1007/978-3-662-53622-3

[41] Colbourn, C. J., and Read, R. C., 1979. "Orderly algorithms for generating restricted classes of graphs". *J. Graph Theory,* ***3***(2), pp. 187–195. doi: 10.1002/jgt.3190030210

[42] Feller, W., 1968. *An Introduction to Probability Theory and Its Applications*, 3rd ed., Vol. 1. Wiley.

[43] PM architectures project. Online. url: https://github.com/danielrherber/pm-architectures-project

[44] Choudum, S. A., 1986. "A simple proof of the Erdos-Gallai theorem on graph sequences". *B. Aust. Math. Soc.,* ***33***(1), Feb., pp. 67–70. doi: 10.1017/S0004972700002872

[45] Godsil, C., and Royle, G., 2001. *Algebraic Graph Theory*. Springer. doi: 10.1007/978-1-4613-0163-9

[46] Brualdi, R. A., and Ryser, H. J., 1991. *Combinatorial Matrix Theory*. Cambridge University Press.

[47] Herber, D. R., and Allison, J. T., 2017. Enhancements to the perfect matching-based tree algorithm for generating architectures. Tech. Rep. UIUC-ESDL-2017-02, Engineering System Design Lab, Urbana, IL, USA, Dec. url: https://hdl.handle.net/2142/98990

[48] Skiena, S. S., 2008. *The Algorithm Design Manual*, 2nd ed. Springer. doi: 10.1007/978-1-84800-070-4

[49] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C., 2009. *Introduction to Algorithms*, 3rd ed. The MIT Press.

[50] Cordella, L. P., Foggia, P., Sansone, C., and Vento, M., 2001. "An improved algorithm for matching large graphs". In Workshop on Graph-based Representations in Pattern Recognition, pp. 149–159.