Published in *Journal of Mechanical Design*, Copyright (c) 2017 by ASME
The final publication is available at ASME Digital Collection via
https://doi.org/10.1115/1.4036132

ENUMERATION OF ARCHITECTURES WITH PERFECT MATCHINGS

Daniel R. Herber*, Tinghao Guo, James T. Allison
University of Illinois at Urbana-Champaign
Industrial & Enterprise Systems Engineering
Urbana, IL 61801
Email: {herber1,guo32,jtalliso}@illinois.edu

Abstract

In this article a class of architecture design problems is explored with perfect matchings. A perfect matching in a graph is a set of edges such that every vertex is present in exactly one edge. The perfect matching approach has many desirable properties such as complete design space coverage. Improving on the pure perfect matching approach, a tree search algorithm is developed that more efficiently covers the same design space. The effect of specific network structure constraints and colored graph isomorphisms on the desired design space is demonstrated. This is accomplished by determining all unique feasible graphs for a select number of architecture problems, explicitly demonstrating the specific challenges of architecture design. With this methodology, it is possible to enumerate all possible architectures for moderate scale-systems, providing both a viable solution technique for certain problems and a rich data set for the development of more capable generative methods and other design studies.

*Corresponding author.

1 Introduction

System architecture is defined as the elements or components contained within a system and their relationships [1–3]. Designing breakthrough engineering systems with new capabilities and new levels of performance requires innovations in system architecture [4, 5]. Engineers often rely on heuristics such as design by analogy [6] and intuition when considering system architecture, but this may result in fixation on example designs and stifle innovation [7].

Consider the following architecture design problem formulation:

$$\min_{\boldsymbol{x}_a} \quad \Phi(\boldsymbol{x}_a) \tag{1a}$$

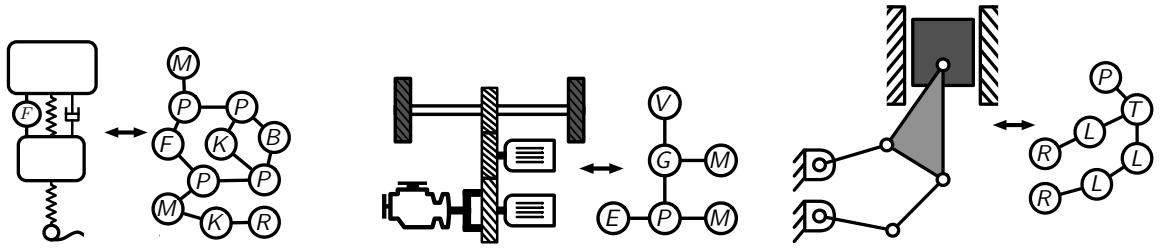
$$\text{subject to:} \quad f_a(\boldsymbol{x}_a) = a \in \mathcal{F}_a \tag{1b}$$

where $\boldsymbol{x}_a$ represents architecture design variables, $f_a(\boldsymbol{x}_a)$ is a mapping between the architecture design variables and the architecture a, and Φ is a performance index. A fair comparison between architecture candidates in the set of feasible architectures $\mathcal{F}_a$ requires knowledge of the best possible performance for each candidate architecture. This requires optimization with respect to *inner-loop* design variables such as plant and control design [8]. The number design variables, constraints, models, etc. can all change depending on the candidate architecture in the inner loop. Hence, formulating and solving architecture design problems can be challenging. Here we focus on a method that generates candidate architectures that may be used to determine the performance index with respect to an architecture-dependent inner-loop design problem.

Many studies have concentrated on effective representation and generation methods, primarily based on graph representations of the system architecture (see Fig. 1 for some common engineering systems represented as graphs). There is a wide range of architecture design representations typically classified along the spectrum of continuum [9, 10] vs. discrete [11, 12] design domains and homogeneous [9, 10] vs. heterogeneous [11, 13] elements. The value of these methods often is to present new valid topologies to engineers for further evaluation (subjective or quantitative), helping to overcome design fixation. A popular class of methods for generating architecture candidates is generative representations [5, 9, 12–17]. This class covers a range of candidate architectures in an implicit form based on repeated application of rules that modify the graph. It has been recognized that generative approaches generate topologically simple designs, not covering the entire design space [11]. Furthermore, the design space is sensitive to design knowledge [6, 16] and rules [9, 17]. While these designs may satisfy functional requirements elegantly, generation of more elaborate architectures is needed in some cases.

It can be challenging to describe the design space of an architecture generation method, partially due to the combinatorial nature of architecture design problems. A better understanding of how certain rules restrict the design space can lead to better generative approaches, but this requires a complete design space to compare against. Furthermore, the ultimate goal is a set of all architectures that are feasible with respect to constraints [18] and that are unique [14]. Arriving at such a design space efficiently is a considerable challenge.

In this article, the design space is completely captured by a perfect matchings approach for a certain class of architecture design problems, more specifically, problems that are represented by undirected colored graphs under the component/port paradigm [2, 11, 19]. The proposed approach generates truly novel architectures (in fact all of them) but still leverages some of the natural



(a) Suspension. **(b)** Hybrid powertrain. **(c)** Mechanism.

FIGURE 1: Architectures represented as graphs.

constraints found in architecture design problems to reduce the number of graphs generated. This design space is found through an enumeration, i.e., a complete, ordered listing of all the items in the collection of feasible and unique architectures. This approach leads to a number of interesting insights into the fundamental nature of architecture design problems.

The remainder of the article is organized as follows. The next section outlines the some of the basic theory behind candidate architectures with perfect matchings. Network structure constraints and the colored graph isomorphism problem are then discussed, with the objective of achieving feasible unique architectures. Using the insights from the previous two sections, a tree search algorithm is developed that more efficiently covers the same design space. A number of case studies are then presented. Finally, a discussion is given of the results and how the proposed approaches can be used in current architecture design research.

2 Candidate Architectures with Perfect Matchings

First, some relevant graph theory background is given.

Definition 1 (Graph) *A graph is a pair $G = (V, E)$ of sets satisfying $E \subset [V]^2$ where the elements of V are the vertices and the elements of E are its edges.*

A *simple graph* is an unweighted, undirected graph containing no graph loops or multiple edges. The adjacency matrix of G is the $n \times n$ matrix $A = A(G)$ whose entries a_{ij} are given by:

$$a_{ij} = \begin{cases} 1 & \text{if the set } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases}$$

For a simple graph, the *adjacency matrix* must have 0s on the diagonal. For an undirected graph, the adjacency matrix is symmetric and if only a subset of the edges are present in E, the correct $A(G)$ can be constructed with $\text{sign}(A + A^T)$. The *connectivity matrix* of simple graph G can be found with:

$$A_C(n) = A^n \tag{2}$$

where the interpretation of $A_C(n)$ is for every nonzero entry, there exists at most n undirected walks required to go from v_i to v_j, i.e., the pair of vertices are connected in some sense [20, p. 165]. We will assume that n is the same as the length of A giving all walks.

The degree of a vertex is the number of edges incident at the vertex and the average degree is $d(G) = 2|E|/|V|$ [21, p. 5]. A *matching* in a graph is a set of edges such that no two have a vertex in common and a perfect matching is a matching that covers every vertex [22, p. 255].

Definition 2 (Colored Graph) *A colored graph G is a three-tuple (V, E, L) where (V, E) specifies an undirected graph and $L = \{V_i\}_{i=1}^{k}$ is a partition of the vertices into color sets ($V_i \cap V_j = \Phi$, $i \neq j$, and $(\cup_{i=1}^{k} V_i) = V$). For convenience, define* $\text{color}(v) = i$ *if* $v \in V_i$.

The graphs in Fig. 1 are colored graphs where each vertex represents a component. The colored labels indicate different component types. For example, in Fig. 1a, (K) represents a vertex with a coloring K indicating that it is a spring component type and that (B) represents a damper component type. These are termed 2-*port* components since they can have up to 2 unique edges if ports are connected to a single additional vertex (this port notion is analogous to bond graph modeling [23]). However, there are fundamental limitations with this representation. For example, if the order that the ports of a component are connected to edges prescribed in A is important, then pure component graph representation is not sufficient for determining a unique architecture. Consider the planetary gear (P) in Fig. 1b. Since the planetary gear is represented by a single vertex, it is unclear which of the connected components {(E), (G), (M)} is connected to the sun, ring, and carrier (common names for the planetary gear ports [24]). Permutations of this decision would result in different architectures but the same adjacency matrix. Another limitation is the unspecified nature of parallel connections when only component-level vertices are used. If the adjacency matrix specifies that K has three connections, it cannot be necessarily uniquely determined what connections are incident at each specific port of K. A better representation would determine unique graphs, motivating a pure ports graph representation of architectures.

2.1 Ports Graph

A port graph G^P is constructed from a three-tuple (C, R, P):
- C is the colored label set representing distinct component types, whose size is denoted by n_C
- R is a column vector indicating the number of replicates for each component type
- P is a column vector indicating the number of ports for each component type

Using (C, R, P) we will create the three-tuple (V, E, L) that defines a proper colored graph (see Definition 2). The definition of an n-port component in this context is all n ports are completely connected to each other. Therefore each component can be considered a complete graph of its ports (see Fig. 2 for the first five complete graphs). The vertex and edge set for G^P is then defined as the union of these complete subgraphs:

$$(V, E)^P = \bigcup_{k=1}^{n_C} \bigcup_{j=1}^{R_k} K_{P_k} \tag{3}$$

where K_{P_k} is a complete graph of size P_k.

The complete label for each vertex is constructed from a naming scheme where the base is the colored label from C, the subscript is the replicate number, and the superscript is the port number.

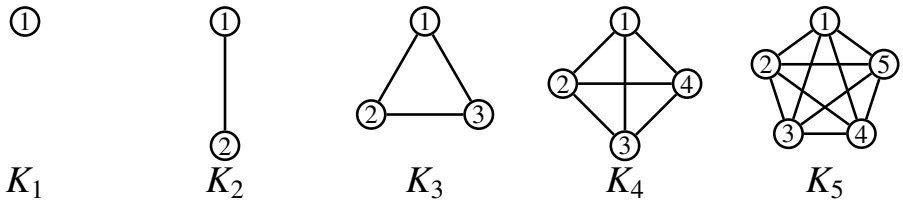


FIGURE 2: Complete graphs on n vertices between 1 and 5.

Then the set of colored labels for G^P can be constructed as:

$$L^P = \bigcup_{k=1}^{n_C} \bigcup_{j=1}^{R_k} \bigcup_{i=1}^{P_k} \left\{ (C_k)_j^i \right\} \tag{4}$$

where each label is unique at this point.

There are a number of graph measures and metrics that can be computed for this class of graphs. First, the number of vertices is given by: $N_p = |V^P| = P^T R$, where $|\cdot|$ is the cardinality of a set. The number of edges in K_n is $n(n-1)/2$ [20, p. 22], so we can easily calculate the number of edges in G^P as:

$$\left|E^P\right| = \frac{1}{2} \left(P \circ (P-1)\right)^T R \tag{5}$$

where $\circ$ denotes the Hadamard product. The total number of components is: $N_C = e^T R$ where e is a column vector of ones of appropriate length.

2.2 Interconnectivity Graph

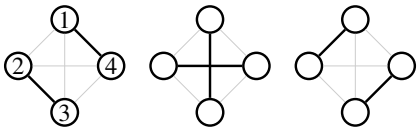
The essence of an architecture design problem is determining the relationships between ports. Therefore a natural question is: what are all the possible architectures? Subgraph enumeration provides a relevant framework for determining all possible graphs satisfying specified properties [22]. We now say that a candidate architecture in an architecture design space described by (C, R, P) has the following properties:

1. A set of components bounded by (C, R, P) (so $n_p \leq N_p$, where n_p is the total number of ports for the candidate architecture).
2. Each port in $(V^P, \{\}, L^P)$, i.e., G^P without edges, is connected to another port (this implies n_p is even and is also known as a complete topology since there are no open ports [19]).

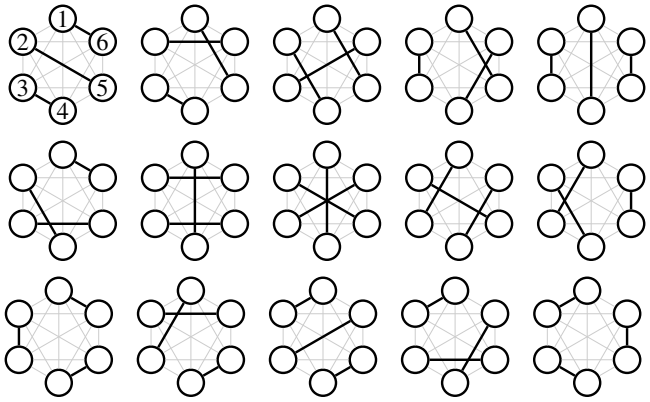
Now, a complete architecture design space described by (C, R, P) would be a set of candidate architectures that contain all possible valid subsets of components and all possible valid edge combinations between said subset of components. We will denote this graph structure space (a set of all graphs that fulfill a certain set of conditions) as $\mathcal{G}_1$. We will now see that the enumeration based on perfect matchings (*PM*s) will correspond to the complete architecture design space.

Recall that a *PM* is a matching in which every vertex of the graph is incident to exactly one edge [22]; all *PM*s for K_2, K_4, and K_6 are shown in Fig. 3. Since a necessary condition for a *PM* is an even number of vertices, we will assume N_p is even (Sec. 3.1.1 discusses the implications of

(a) 1!! perfect matchings for K_2.



(b) 3!! perfect matchings for K_4.



(c) 5!! perfect matchings for K_6.

FIGURE 3: Perfect matchings for K_2, K_4, and K_6.

this restriction). The number of *PM*s for K_n can be calculated using the double factorial function:

$$\mathcal{D}(n) = (n-1)!! = (n-1) \times (n-3) \times \cdots \times 3 \times 1 \quad n \text{ even} \tag{6}$$

and the first several values of this function are $\mathcal{D}(2) = 1$, $\mathcal{D}(4) = 3$, $\mathcal{D}(6) = 15$, $\mathcal{D}(8) = 105$, $\mathcal{D}(10) = 945$, $\mathcal{D}(12) = 10,395$, $\mathcal{D}(14) = 135,135$, $\mathcal{D}(16) = 2,027,025$, $\mathcal{D}(18) = 34,459,425$ [25]. This function grows slower than the traditional factorial function since the even elements have been omitted. This result agrees quite well with the bound by Mittal and Frayman for a similar problem (the bound being on order of $\sqrt{N_p!}$) [2].

For n_m between 1 and $\mathcal{D}(N_p)$, $\mathcal{P}(n_m, N_p)$ denotes the edge set for the n_mth *PM* of K_{N_p}. The uniqueness of each *PM* can be ensured by ordering all edges with the first element being the larger vertex (in the sense of the index value). For example for the first graph in Fig. 3b, $\mathcal{P}(1,4) = \{(4,1),(3,2)\} \neq \{(1,4),(2,3)\}$. It will be convenient to map the edge set to a vector where sequential pairs are a single edge (e.g., $\{(4,1),(3,2)\} \rightarrow$ [4 1 3 2]. There are two interesting properties of the set of all *PM*s [22]:

1. Enumerating all *PM*s of K_N results in a set of graphs where all possible edge combinations are present where each port is connected to exactly one other port.
2. The set of *PM*s graphs of K_N, contains all edge sets for K_{N-2}, where $N \geq 4$.

Now with these two properties in mind, consider any candidate architecture defined by the properties above. It has $n_p \leq N_p$ and n_p is even, so there exists a *PM* matching edge set in the set of *PM* graphs of K_{N_p} that contains the subset of components and the particular edge combination since all edge combinations of n_p are found in the set N_p. Therefore, because any candidate architecture

ALGORITHM 1: Creation of edge set for a specific perfect matching number.

Input : N – number of vertices (should be even)
I – perfect matching number, integer between 1 and $(N-1)!!$
Output: E – vector of edges in sequential pairs

```
J ← [1, 3, 5, ..., N − 1]                                   /* odd numbers from 1 to N-1 */
P ← [1, cumprod(J)]                                         /* cumulative double factorial */
V ← [1, 2, ..., N]                                          /* create initial list of available vertices */
for j ← J do
    q ← (N + 1 − j)/2                                       /* index for 2nd to last entry in P */
    I ← ceil(I/P(q))                                        /* calculate smaller vertex index */
    E(j) ← v(end)                                           /* assign largest remaining value */
    remove element V(end)                                   /* remove largest remaining value */
    E(j + 1) ← V(i)                                         /* assign smaller selected value */
    remove element V(i)                                     /* remove the smaller selected value */
    I ← I − ((i − 1) × P(q))    /* subtract to get index in subgraph with 2 vertices removed */
end
```

can be found in the set of PM graphs of K_{N_p}, a PM approach captures the complete architecture design space.

A PM approach is a type of *graph numerical representation scheme* (GNRS) since there is a binary relation between $\mathcal{G}_1$ and $n_m \in [1, \mathcal{D}(N_p)]$. A PM approach is left-total and left-unique with respect to complete topologies of (C, R, P) (left implies a map from $\mathcal{G}_1$ to n_m and these are desired properties for a GNRS) [3]. Algorithm 1 is useful for this direction as it determines $\mathcal{P}(n_m, N_p)$ [22, 26][1]. In addition, a PM approach is right-total and right-unique with respect to the same conditions (right implies a map from n_m to $\mathcal{G}_1$). Algorithm 2 is useful for the right direction as it determines $\mathcal{P}^{-1}(E)$ where E is a valid PM edge set. Based on these two efficient algorithms, a PM approach is *algorithmic* in both directions [3].

The interconnectivity graph G^I then is defined with V^P, L^P, and an edge set from the set of PMs:

$$E^I = \mathcal{P}(n_m, N_p) \qquad \text{where } n_m \in \left[1, 2, \ldots, \mathcal{D}(N_p)\right] \tag{7}$$

The number of edges is: $|E^I| = N_p/2$.

2.3 Connected Ports/Components Graph

The connected ports graph is the union of the ports graph and interconnectivity graph:

$$G^{CP} = G^P \cup G^I \tag{8}$$

The number of vertices is still N_p. There is possibility of multiple edges when combining the graphs since edges between the already connected ports of a component may be connected with a PM. We can simplify G^{CP} by combining all multiple edges into a single equivalent edge, thus

[1] Ref. [26] contains MATLAB codes for Alg. 1 and more efficient recursive algorithm when all perfect matchings are required.

ALGORITHM 2: Determination of the perfect matching number for a specific edge set.

Input : E – vector of edges in sequential pairs, should be properly ordered and even length
Output: I – perfect matching number, integer between 1 and $(N-1)!!$

$N \leftarrow$ `length`(E) /* total number of vertices */
$P \leftarrow$ reverse elements of `cumprod`$([1,3,5,\cdots N-3])$ /* array flip, cumulative double factorial */
$V \leftarrow [1,2,\ldots,N]$ /* create initial list of available vertices */
$I \leftarrow 1$ /* initialize perfect matching index */
for $j \leftarrow 1$ **to** $N/2-1$ **do**
    $q \leftarrow E(2j-1)$ /* index of largest remaining vertex */
    $V(q) \leftarrow 0$ /* zero entry, removing it */
    $i \leftarrow$ find index of $E(2j)$ in the vector of nonzero elements of V
    $V(E(2j)) \leftarrow 0$ /* zero entry, removing it */
    $I \leftarrow I + ((i-1) \times P(j))$ /* sum to get build up the index */
end

creating a simple graph. Using this operation, the number of edges of G^{CP} can be bounded by:

$$\left|E^P\right| \leq \left|E^{CP}\right| \leq \left|E^P\right| + \left|E^I\right| \tag{9}$$

since each edge of E^I could be a repeat of an edge in E^P. G^{CP} is a unique representation of an architecture since a *PM* is between specific ports and all components are fully connected subgraphs.

There are advantages of the component graph representation that should be utilized, such as a reduced number of vertices and edges (and not differentiating replicates). We now characterize *simple components* whose port ordering does not matter (e.g., a 2-port spring) and *structured components* where it does (e.g., a 3-port planetary gear). All simple components will be reduced to a single vertex and the appropriate edges will be created. The labels for simple components will be modified by removing both the superscript and subscript of L^{CP}. Structured components will only have their subscripts removed to maintain port discernibility. This graph representation is termed the connected component graph G^{CC}.

To get a better sense of the structure of G^{CC} consider all components to be simple components. Then the number of vertices is simply N_C. The number of edges of G^{CC} can be bounded by $0 \leq |E^{CC}| \leq |E^I|$ and the labels are:

$$L^{CC} = \bigcup_{k=1}^{n_C} \bigcup_{j=1}^{R_k} \{C_k\} \tag{10}$$

All graphs in the remaining sections are considered to be G^{CC} graphs unless otherwise noted.

3 Candidate Graphs to Unique Useful Graphs

In the previous section an approach for enumerating architectures based on the consideration of every potential *PM* was outlined. However, there are a number of deficiencies in this set of architectures, including infeasible graphs based on practical constraints for a specific architecture design problem, and repeated graphs in the modeling sense. The following two subsections ad-

dress architecture feasibility and uniqueness, respectively.

3.1 Network Structure Constraints

We define feasibility as a candidate architecture's satisfaction of all network structure constraints (NSCs) for a particular architecture design problem. Similar to the rules in generative design approaches, the creation of the set of network structure constraints for a particular architecture design problem is subject to the creativity and intuition of the designer. Some of these constraints may be fairly self-evident while others might be vague or contentious (Ref. [18] discusses these issues). Wyatt et al. describe four types of NSCs that are sufficient to define almost all aspects of realizability of an architecture and are summarized briefly (without edge coloring considerations) as [18]:

- *Component number constraints* (CNCs) prescribe how many components of a given type must be present
- *Direct connection constraints* (DCCs) prescribe which component types may be connected together by which connection types and cardinality of the connections
- *Fan out constraints* (FOCs) prescribe how many connections that components of a certain type must have in total
- *Indirect connection constraints* (ICCs) prescribe how many continuous paths there must be from every component of one type to every component of another type

Graph generating algorithms designed to always satisfy certain NSCs could be more useful since all of the generated graphs would be feasible with respect to those certain NSCs. Furthermore, graph generating algorithms can also be designed to produce graphs that have a higher proportion that satisfies certain NSCs than more naive approaches. Some of the NCSs that are not satisfied can be with edits to the graph. The only operation that will be considered here is the removal of vertices G^{CC} and the corresponding edges and labels as it will maintain certain properties of the graph structure space $\mathcal{G}_1$. Other operations such as vertex insertion, edge insertion, or label substitution destroy the analysis of the design space coverage that is possible with an enumerative *PM* approach. Next, several common NSCs are described along with the specifics of checking their satisfaction with available graph analysis tools.

S_1 Every graph must be a connected graph (ICC). A graph is termed connected if there is a path from any vertex to any other vertex in the graph [21, p. 18]. This can be checked with the connectivity matrix, $A_C(G)$, in Eqn. (2). If all entries in this matrix are not 1, then the graph is not connected.

S_2 Every graph can only have a maximum number of a given component type (CNC). This is defined by R in the architecture definition three-tuple so is naturally handled by a *PM* approach. An example: 'Every suspension must have less than 3 springs'.

S_3 Every graph must have a specific number of certain component types (CNC). These mandatory components will be captured with a vector M of length n_C. The elements of M are binary with a 1 indicating all replicates of the component type must be present in the graph. An example: 'Every hybrid powertrain must have an engine and a vehicle'.

S_4 Every graph must have specific component types connected to each other (ICC). This can be checked with the connectivity matrix in Eqn. (2). If nonzeros are not present at every location where a path must exist between component types, then the graph is infeasible. If we require S_1 and S_3, then we can leverage the vector M in S_3 to satisfy both constraints by checking $A_C(G)$ such that all mandatory components are connected to each other. An example: 'Every hybrid powertrain must have an engine connected to a vehicle'.

S_5 Every graph must have vertices whose number of unique edges is within a specific range (FOC). The values in P can define the upper bound for each vertex since components are defined by a certain number of ports. For even port numbered component types the lower bound is 0 and 1 for odd. This can be checked summing row-wise (or column-wise) the symmetric adjacency matrix $A(G)$ and comparing these sums to the appropriate index in P. This type of NSC is sometimes termed a degree-constrained subgraph problem [27, p. 217]. A *PM* approach naturally satisfies this constraint. An example: 'Every spring must have between 0 and 2 unique edges'.

S_6 Every graph must have vertices with a specified number of unique connections (FOC). This is a stronger form of S_5 where both the upper and lower bound can be determined by P and is sometimes termed a factors problem [27, p. 218]. An example: 'Every spring must have exactly 2 unique edges'.

S_7 Every graph must have edges between vertices that are feasible (DCC). We can specify that certain component types cannot be connected to other component types with a reduced potential adjacency matrix A_R. This $n_C \times n_C$ binary matrix will have 1 entries indicating a connection is feasible and 0 entries for infeasible. This constraint can be checked by verifying that each 1 in $A(G)$ has a corresponding 1 in the potential adjacency matrix. No self-loops in a specific component type can be enforced with a 0 at the appropriate location on the diagonal of A_R. A *PM* approach does not satisfy this constraint as all connections between ports are considered feasible. An example: 'Every translational spring cannot be connected to any rotational damper'.

The ordering of the constraints matters if vertices are to be removed to satisfy certain constraints. The following procedure is assumed:

0. S_2 and S_5 naturally satisfied with a *PM* approach
1. Check S_3 and S_4 simultaneously using M since they can be checked without needing to remove the removable components
2. Remove components that don't satisfy S_3 and S_4; thus satisfying S_1
3. Check S_6
4. Check S_7
5. Check any other constraints

The specific steps are only performed if the constraint is present in a specific architecture design problem. The ordering of S_6, S_7, and the other previously undefined constraints could be performed in an alternative order as long as they are checked after removable components are removed. This is so that a candidate architecture is not declared infeasible if only removed components and their connections violate the constraints.

The graph structure space defined as graphs that satisfy the present NSCs and (C, R, P) is denoted $\mathcal{G}_2 \subseteq \mathcal{G}_1$. The NSCs $\{S_1, S_3\ S_4\}$ are assumed to be all present or none present to simplify the discussion as many common architecture design problems require all three. With the NSCs outlined, a *useful graph* is defined as one that is feasible with respect to the NSCs.

3.1.1 Comparison to Another Method

At this point, it is imperative to compare the *PM* approach to another graph numerical representation scheme that can be used for enumeration: Indexed Stacked Blocks (ISBs) [3]. This scheme is far more general than the proposed *PM* approach as it allows for directed graphs, edge coloring, enumeration of potential colored label sets, and variable number of nodes. All permutations of the candidate adjacency matrices are considered. The graph structure space for the ISB approach is denoted $\mathcal{G}_0$ since $\mathcal{G}_1 \subseteq \mathcal{G}_0$. However with this generality comes an enormous space, potentially too large to be useful for certain problems.

We can analyze this statement by observing how the ISB block method handles some of the proposed NSCs. For a fair discussion, we should restrict the space to a certain block (fixed number of vertices and color label set ordering). Then both S_2 and S_7 can be naturally satisfied by removing the infeasible entries in the adjacency matrix. However, S_5 is not satisfied for large portions of $\mathcal{G}_0$; the degree of a vertex is not directly controlled. Once additional NSCs are added, the probability that an index results in a feasible graph might be so small that none are ever found.

To illustrate this consider the number of permutations of $A(G^{CC})$ with N_C components [3]:

$$\mathcal{T}(N_c) = 2^{N_c(N_c-1)/2} \tag{11}$$

with the first several values being $\mathcal{T}(1) = 1$, $\mathcal{T}(2) = 2$, $\mathcal{T}(3) = 8$, $\mathcal{T}(4) = 64$, $\mathcal{T}(5) = 1,024$, $\mathcal{T}(6) = 32,768$, $\mathcal{T}(7) = 2,097,152$, $\mathcal{T}(8) = 268,435,456$. Now consider the case when $N_p = 30$ and $N_c = 20$, then there are 6×10^{15} *PM*s versus 2×10^{57} adjacency matrix permutations. Both numbers are quite large but a clear combinatorial advantage is seen with the *PM* approach (see Fig. 4). This will be exacerbated when structured components considered. However, since N_p and N_c can be different, there are some combinations where $\mathcal{T}(N_c)$ is actually smaller than $\mathcal{D}(N_p)$. This is shown in the figure with the curved line $\mathcal{D}(N_c) = \mathcal{T}(N_p)$. Most architecture design problems are above this line.

A *PM* approach can be seen as an alternative to permuting all possible adjacency matrices assuming the architecture design problem is based on (C, R, P) with NSC S_5. The question then becomes does every port being filled as in a *PM* approach result in all architectures defined by a certain architecture design problem? Consider that we can always include 1-port components that represent empty connections, i.e., this component type implies that the vertex and edge can be removed from the graph without loss. We can control what components are allowed to have empty connections with S_7. Certain NSC sets such as $\{S_1, S_3\ S_4, S_6\}$ would also require every port to be filled.

3.2 Colored Graph Isomorphisms

If we have a list of useful graphs, how many of them are truly different? Determining if two graphs are 'different' is known as the graph isomorphism problem. We define uniqueness among a set of

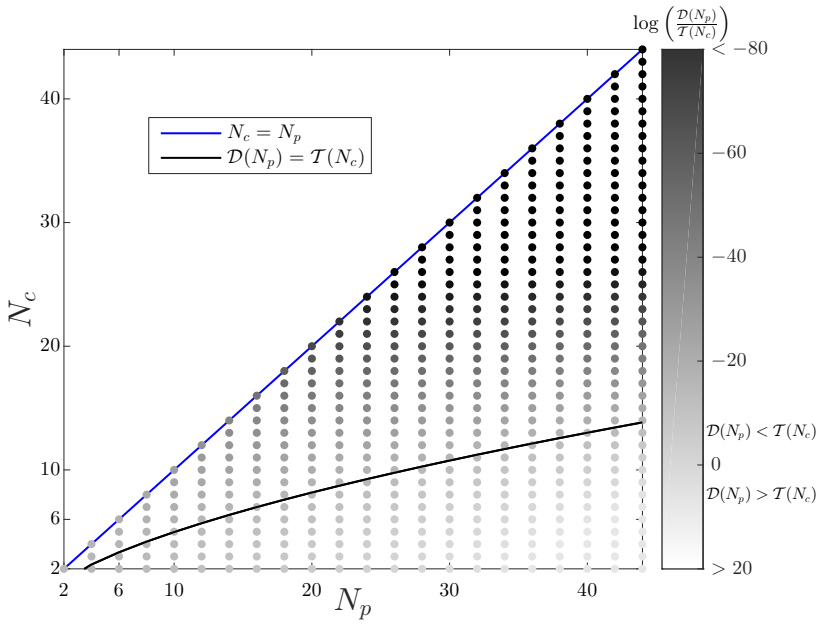


FIGURE 4: Comparison of number of graphs with *PM* approach and adjacency matrix approach.

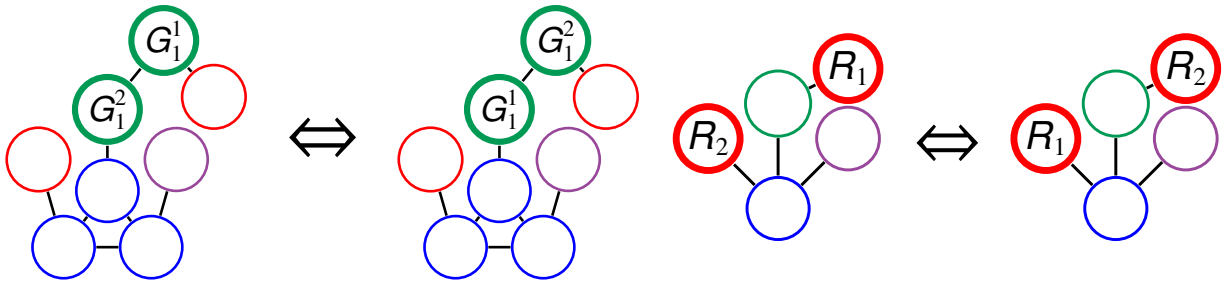
architecture graphs to mean that no two candidate architectures are isomorphic.

Definition 3 (Isomorphism) *Let $G = (V, E)$ and $G' = (V', E')$. We call G and G' isomorphic, and write $G \simeq G'$, if there exists a bijection $\phi : V \to V'$ with $(v_i, v_j) \in E \Leftrightarrow (\phi(v_i), \phi(v_j)) \in E'$ for all $v_i, v_j \in V$. The map ϕ is called an isomorphism [21].*

Definition 4 (Colored Graph Isomorphism) *The colored graph isomorphism problem is to decide the existence of a color preserving isomorphism between a pair of colored graphs $G = (V, E, P)$ and $G' = (V', E', P')$, i.e., a mapping $\phi : V \to V'$ satisfying the following conditions:*
1. ϕ is an isomorphism by Definition 3.
2. $\text{color}(v) = \text{color}(\phi(v))$ *for all $v \in V$.*

We can better understand how the colored graph isomorphism problem affects the architecture design problem by looking at two different isomorphisms:

- *Port-type isomorphism* occurs when a component has ports that are indistinguishable in a modeling sense and can occur when using a ports representation. We have already termed such components as simple components. For example, consider a 2-port component that physically represents a mechanical translational spring. The two ports can be permuted and the resulting physical model will be equivalent. This demonstrated in Fig. 5a with the simple component type G. G^{CC} for the same graphs would be identical since the information about specific ports is lost. We leverage this fact to perform an *initial port-type isomorphism filter* to remove *PM*s that certainly have a port-type isomorphism. For a given simple n-port component, there are $n!$ ways to arrange the ports such that a port-type isomorphism occurs.
- *Component-type isomorphism* occurs when switching a pair of component type replicates preserves the graph. This type of isomorphism is present due to the arbitrary subscript numbers assigned to each vertex and is demonstrated in Fig. 5b. The 1-port component type R is permuted but since R_1 and R_2 are the same component type, the graph remains the same (in the



(a) Port-type. **(b)** Component-type.

FIGURE 5: Two different type of isomorphisms.

sense of Definition 4). For n replicates of a component type, there are $n!$ ways to arrange the components such that a component-type isomorphism occurs.

We now define the final graph structure space $\mathcal{G}_3 \subseteq \mathcal{G}_2$ representing all unique useful graphs. Assuming no NSCs except those naturally satisfied by a *PM* approach, we can discern a very rough lower bound on the size of this set with:

$$\mathcal{D}(N_p) \times \prod_{i=1}^{n_C} \frac{1}{R_i! \times (P_i!)^{R_i}} \leq |\mathcal{G}_3| \leq \mathcal{D}(N_p) \tag{12}$$

where this formula assumes all port-type and component-type isomorphisms that could occur, do occur in the set of *PM*s. Consider $(C, R, P) = (A, 6, 1)$, then Eqn. (12) provides a lower bound of 0.02 graphs, but we know there is exactly 1 unique graph.

Although the graph isomorphism problem is NP, there are many efficient practical algorithms [28]. Study of the graph isomorphism problem is an ongoing field and recent breakthroughs could lead to improved algorithms [29]. In this work, we utilize the python package igraph using the `isomorphic_vf2` function [30] based on the VF2 algorithm [31] to solve the colored isomorphism problem.

Many architecture design studies ignore the isomorphism problem but presence of isomorphic graphs leads to the evaluation of non-unique options [14]. For certain problem sizes, the complexity of checking for isomorphisms may be much greater than generating and evaluating new, potentially non-unique graphs. But to understand the effect of problem definition, NSCs, and candidate graph generation algorithms on $\mathcal{G}_3$ requires the isomorphism checks, and can lead to insights into new algorithms that naturally avoid the isomorphism problem [32]. Other graph generation algorithms have been developed that avoid isomorphic graphs [33, 34]. Furthermore, for appropriately sized problems, the isomorphism check is computationally viable.

Algorithm 3 was developed to determine $\mathcal{G}_3$ given $\mathcal{G}_2$. This algorithm checks a candidate graph against bins of already found unique graphs and stops checking if an isomorphism is found, making it parallelizable to a degree and removes unnecessary checks. There are a number of quick preliminary checks that can be done between two graphs, as necessary conditions for them to be isomorphic include having the same number of vertices, edges, and color label distributions.

ALGORITHM 3: Determination of the unique colored graphs given a set of colored graphs.

Input : *Graphs* – set of colored graphs
Nbin – number of bins (for parallel processing)
Output: *UniqueGraphs* – set of unique colored graphs

```
1  ind ← 1                                          /* initialize index for total unique graphs */
2  bin(1).Graphs(1) ← Graphs(1)                     /* first graph is always unique */
3  for i ← 2 to length(Graphs) do                   /* check remaining graphs */
4      G1 ← Graphs(i)                               /* current graph to check */
5      for j ← 1 to min(Nbin, ind) do in parallel   /* check against each nonempty bin */
6          k ← length(bin(j).Graphs)                /* unique graphs in bin */
7          IsoFlag ← 0                              /* initialize flag, 0 is not isomorphic */
                                                    /* while graphs remain and isomorphism not found */
8          while (k > 0) and (IsoFlag = 0) do
9              G2 ← bin(j).Graphs(k)                /* a unique graph */
10             if G1 and G2 pass preliminary isomorphism checks then
11                 IsoFlag ← isomorphic_vf2(G1, G2) /* return 1 if G1 and G2 are isomorphic
                       */
12             end
13             k ← k − 1                            /* decrease index since G2 checked */
14         end
15         results(j) ← IsoFlag                     /* assign result for bin c */
16     end
17     if all elements of results are 0 then        /* if no isomorphisms */
18         J ← mod(ind, Nbin) + 1                   /* index for next smallest bin */
19         bin(J).Graphs(end + 1) ← G1              /* assign to a bin */
20         ind ← ind + 1                            /* total unique graphs */
21     end
22 end
23 UniqueGraphs ← combine graphs in bin into a single set of graphs
```

4 Tree Search Algorithm

With a better understanding of the colored graph isomorphism problem in the context of architecture design, a tree search algorithm was developed to more efficiently enumerate a graph structure space that contains $\mathcal{G}_3$. This algorithm is based on the idea that for simple components, the port ordering does not matter so we are free to always choose the first port of a component when making edges.

Algorithm 4 starts with a vector for length N_C where the entries are the number of ports for every component in G^{CC}. For example, if P = [1 2] and R = [2 3], then this vector would be V = [1 1 2 2 2] and cp = [2 3 5 7 9]. Recursion is then applied to enumerate all possible edge combinations where each recursive step adds an edge. The end result is a set of missorted *PM*s, i.e., the sequential pairs that define the edges need to be sorted such that they fit the definition of a *PM* in Sec. 2.2 (but no *PM* will occur twice and the property of naturally satisfying S_2 and S_5 is maintained). The end result is an algorithm that does not produce many *PM* graphs that would certainly have a port-type isomorphism. A visualization of the tree-like behavior is in Fig. 6 where each leaf in the tree is a new call of the algorithm and the branches are the loops through the possible remaining edges. Not all leaves have the same number of branches since components

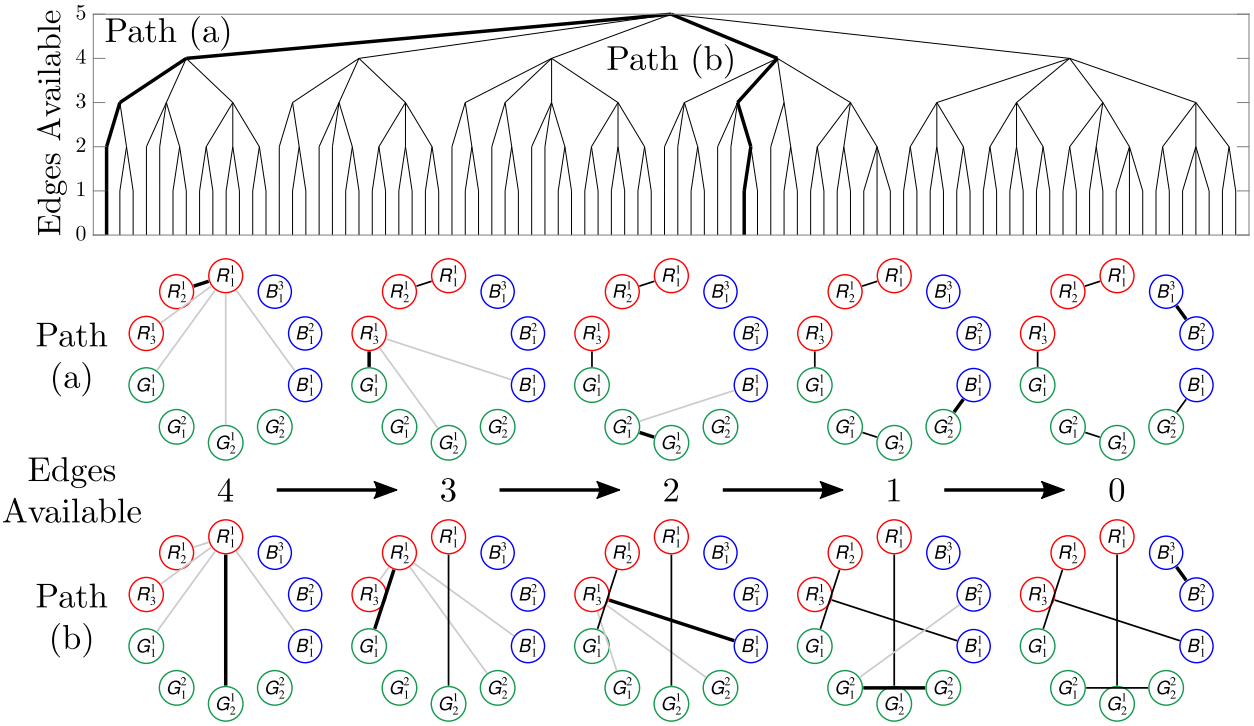


FIGURE 6: Tree structure for Case Study 1 using the basic tree search algorithm in Alg. 4.

become completely connected at different times. Two paths are shown in the figure for a particular architecture design problem, each resulting in a different *PM*. Note that the thicker black lines indicate the chosen edge and the gray lines are the other potential edges for the particular leaf instance.

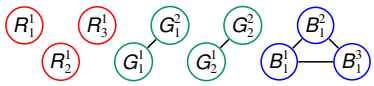
This approach is similar to a number of reported graph enumeration algorithms [19, 34–36]. The tree algorithm shares some similarities with deletion orderly algorithms [36]. Snavely and Papalambros developed a tree algorithm that only deals with structured components, which limits the design space by the number of components rather than the (C, R, P) notation in this work [19]. Faulon et al. enumerate molecules of a specific signature height with a recursive algorithm, but do not allow for components to be removed. Thus, they do not cover the same architecture design space [34]. Neither of these works make the connection between *PM* theory and architecture enumeration which provides a number of insights and practical functions.

We can further improve on this algorithm by adding a single line between lines 3 and 4 that will result in graphs that always satisfy S_7 (feasible edge constraints). First, expand A_R such that its size is the same as G^{CC} where 0 entries still indicate infeasible edge constraints. The additional line would then be: $AV \leftarrow A(I(1),:) \circ V$. By finding the nonzero entries of AV instead of V, we limit the for-loop to edges that are feasible. This has the intentional effect that certain branches of the tree will terminate before a feasible *PM* is found. Therefore when S_7 is present, we will utilize this 'improved' tree search algorithm to more efficiently enumerate $\mathcal{G}_3$.

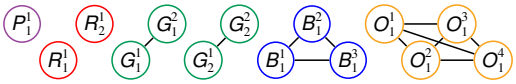
ALGORITHM 4: Basic tree search algorithm.

Input : M – set of graphs, start empty
V – vector of remaining ports for each component
E – vector of edges in sequential pairs, start empty
cp – cumulative sum of the original V plus 1
Output: M – set of graphs

```
1  I ← find(V)                                          /* find nonzero entries */
2  L ← cp(I(1)) – RP(I(1))                              /* left port */
3  V(I(1)) ← V(I(1)) – 1                                /* remove port */
4  I ← find(V)                                          /* find nonzero entries */
5  for i ← I do                                         /* loop through all nonzero entries */
6      V2 ← V                                           /* local remaining ports vector */
7      R ← cp(i) – V(i)                                 /* right port */
8      E2 ← [E, L, R]                                   /* combine left, right ports for an edge */
9      V2(i) ← V2(i) – 1                                /* remove port (local copy) */
10     if any element of V2 is nonzero then
11         M ← Algorithm 4 with M, V2, E2, cp
12     else
13         M{end + 1} ← E2                              /* missorted perfect matching */
14     end
15 end
```



(a) Case Study 1.



(b) Case Study 2.

FIGURE 7: G^P graphs for two examples.

5 Enumeration Case Studies

In this section, a number of case studies are provided to demonstrate the theoretical aspects of the previous sections[2].

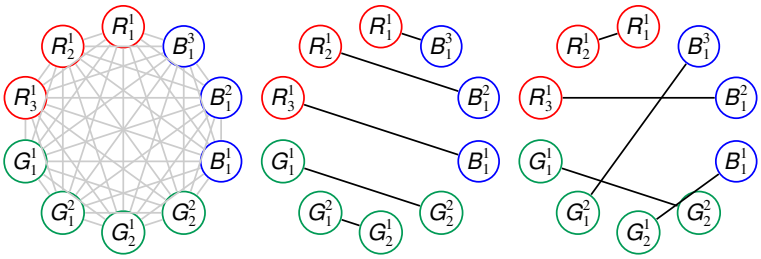
5.1 Case Study 1

The base three-tuple is specified as:

$$C = \{R, G, B\}, \qquad R = [3, 2, 1]^T, \qquad P = [1, 2, 3]^T$$

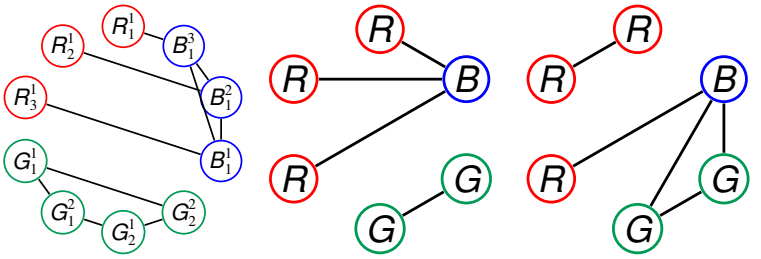
This example has three different simple component types that have multiple ports and replicates

[2]Ref. [37] contains MATLAB codes that replicate the results from these enumeration case studies and can generate graphs for (C, R, P) architecture design problems.



(a) K_{10}. **(b)** *PM* 1. **(c)** *PM* 462.

FIGURE 8: Select interconnectivity graphs for Case Study 1.



(a) G^{CP} for *PM* 1. **(b)** G^{CC} for *PM* 1. **(c)** G^{CC} for *PM* 462.

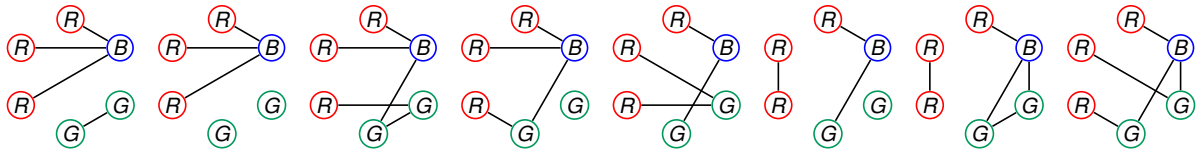
FIGURE 9: Select connected ports and connected component graphs for Case Study 1.

and is visualized in Fig. 7a. Then G^P is:

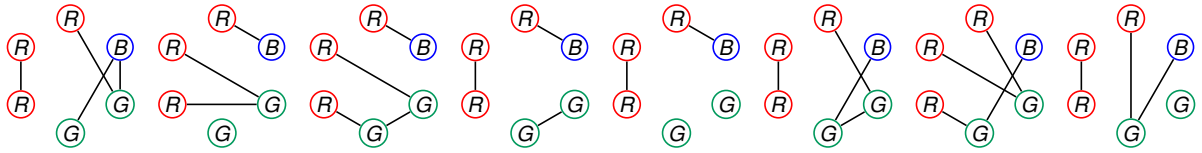
$$
\begin{aligned}
V^P &= \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\} \\
E^P &= \{(4, 5), (6, 7), (8, 9), (8, 10), (9, 10)\} \\
L^P &= \left\{R_1^1, R_2^1, R_3^1, G_1^1, G_1^2, G_2^1, G_2^2, B_1^1, B_1^2, B_1^3\right\} \\
N_p &= 10, \quad N_c = 6, \quad 3.28 \leq |\mathcal{G}_3| \leq 945
\end{aligned}
$$

In Fig. 8 we see G^I for two different *PM*s. Then in Fig. 9a we can see G^{CP} for *PM* 1. This can then be mapped to the equivalent G^{CC} shown in Fig. 9b. The basic tree search algorithm will have the same tree regardless of the NSCs and this tree is visualized in Fig. 6. Both graphs in Figs. 9b and 9c will have a topologically equivalent graph appear in the set of graphs generated by this algorithm. Now two different sets of NSCs will be discussed.

1. *No additional NSCs.* There are 32,768 adjacency matrices ($\mathcal{G}_0$); 945 *PM*s ($\mathcal{G}_1$); 86 candidate graphs with basic tree search algorithm; 77 remaining candidate graphs after initial port-type isomorphism filter; and 77 feasible graphs ($\mathcal{G}_2$). Finally, there are only 16 unique graphs ($\mathcal{G}_3$) consistent with Eqn. (12) and all shown in Fig. 10. 258 graph comparisons were needed and only 113 required a full isomorphism check.
2. *NSCs S_1, S_3 and S_4 with M = [0 0 1], and S_6 with P as the number of unique edges.* Same as 1 above until the feasibility checks and here there are only 23 feasible graphs due to the additional NSCs. Finally, there are only 5 unique graphs ($\mathcal{G}_3$) all shown in Fig. 11. Note that vertices not connected to (B) have been removed. 37 graph comparisons were needed and only 19 required a full isomorphism check. If M = [1 1 1], then only 2 unique designs are possible.

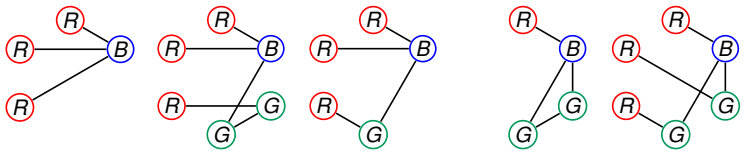


(a) *PM* 227 **(b)** *PM* 228 **(c)** *PM* 332 **(d)** *PM* 438 **(e)** *PM* 467 **(f)** *PM* 468 **(g)** *PM* 678 **(h)** *PM* 691



(i) *PM* 700 **(j)** *PM* 844 **(k)** *PM* 850 **(l)** *PM* 852 **(m)** *PM* 855 **(n)** *PM* 895 **(o)** *PM* 904 **(p)** *PM* 913

FIGURE 10: All 16 unique graphs with no additional NSCs for Case Study 1.



(a) *PM* 227 **(b)** *PM* 332 **(c)** *PM* 438 **(d)** *PM* 678 **(e)** *PM* 691

FIGURE 11: All 5 unique graphs for Case Study 1 requiring a connected graph containing B and a specified number of unique edges.

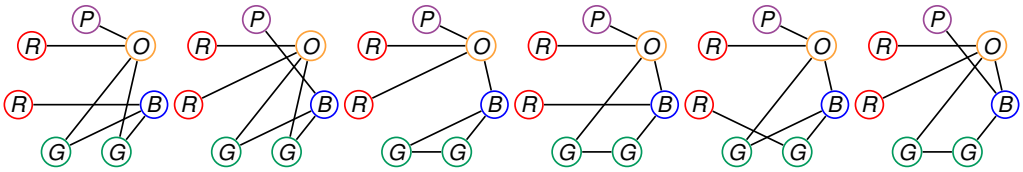
5.2 Case Study 2

This example has five different component types that have multiple ports including multiple 1-port component types and is visualized in Fig. 7b. The base three-tuple is specified as:

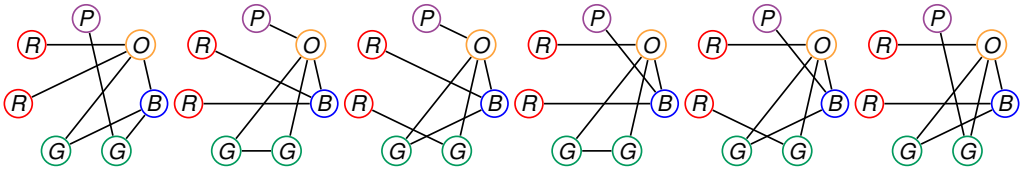
$$C = \{P, R, G, B, O\}, \quad R = [1, 2, 2, 1, 1]^T, \quad P = [1, 1, 2, 3, 4]^T$$
$$N_p = 14, \quad N_c = 7, \quad 58.7 \leq |\mathcal{G}_3| \leq 135135$$

1. *No additional NSCs.* 2,097,152 adjacency matrices ($\mathcal{G}_0$); 135,135 *PM*s ($\mathcal{G}_1$); 1,119 candidate graphs with basic tree search algorithm; 767 remaining candidate graphs after initial port-type isomorphism filter; 767 feasible graphs ($\mathcal{G}_2$); 260 unique graphs[3] ($\mathcal{G}_3$). 41,036 graph comparisons were needed and only 11,828 required a full isomorphism check.
2. *NSCs S_1, S_3 and S_4 with M =* [1 0 0 0 0]. Same as 1 above except there are only 137 unique graphs that contain (P) and are connected.
3. *NSCs S_1, S_3 and S_4 with M =* [1 1 1 1 1]*, and S_6 with P as the number of unique edges.* Same as 1 above until the feasibility checks and here there are only 31 feasible graphs due to the additional NSCs. Finally, there are only 12 unique graphs all shown in Fig. 12. 102 graph comparisons were needed and all 102 required a full isomorphism check.
4. *Same as 3 except with $C = \{P, R, G, G, B, O\}$ modified, M =* [1 1 1 0 1 1] *, and appropriate changes to P and R.* Therefore requiring at least one (G) to be present but otherwise the same. Now there are 34 feasible graphs and 14 are unique (more than 3 since this is a less constrained problem).

[3]The 260 graphs can be seen at http://systemdesign.illinois.edu/publications/IDETC2016-60212/study2graphs.php



(a) *PM* 54923 **(b)** *PM* 55043 **(c)** *PM* 95560 **(d)** *PM* 96505 **(e)** *PM* 96506 **(f)** *PM* 96625



(g) *PM* 96626 **(h)** *PM* 99544 **(i)** *PM* 99547 **(j)** *PM* 99559 **(k)** *PM* 99562 **(l)** *PM* 99563

FIGURE 12: All 12 unique graphs for Case Study 2 requiring all components to be connected and a specified number of unique edges.

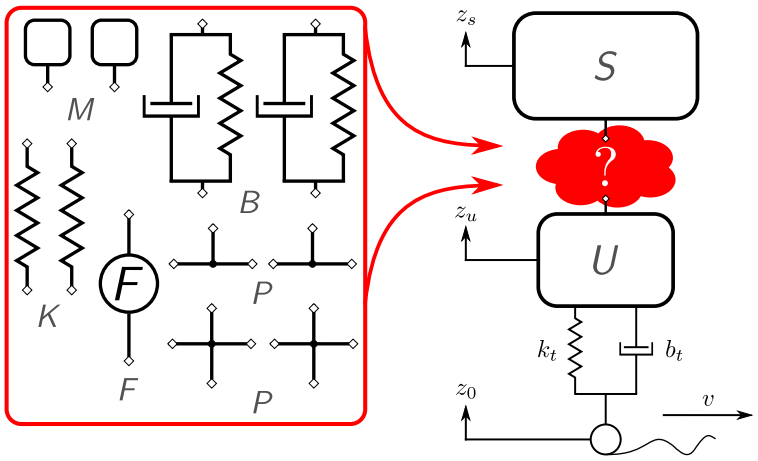


FIGURE 13: Suspension architecture enumeration case study.

5.3 Case Study 3

A graph representing a quarter-car suspension was introduced in Fig. 1a; now we will seek graphs that have different topologies between the sprung (S) and unsprung (U) masses represented in Fig. 13. These graphs could then be used with design studies that evaluate the performance of a particular suspension architecture (see Ref. [38] for a design study on a particular architecture). The components considered will be additional masses, springs, dampers, a force actuator, and parallel connections (these are schematically shown in Fig. 13). The specific selection of (C, R, P) was chosen to be near the limits of what is currently possible with the proposed enumeration methods in this article.

Some additional assumptions are also made on the component definitions. First, B is a parallel damper and spring to ensure that there is a stable equilibrium point for the damper. Next, both 3- and 4-port parallel components are included to facilitate more efficient generation of the useful architectures. A 4-port parallel connection is equivalent to two 3-port parallel connections but the 4-port component provides structure that can be utilized with some specific NSCs. With the

problem outlined, the base three-tuple is specified as:

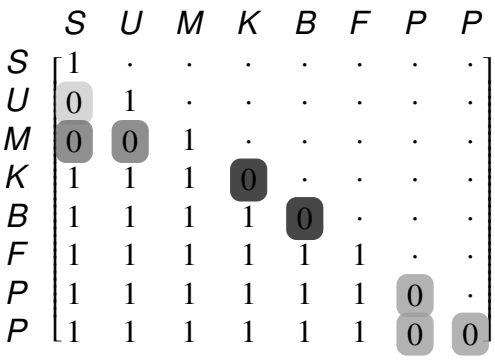
$$C = \{S, U, M, K, B, F, P, P\}, \quad R = [1, 1, 2, 2, 2, 1, 2, 2]^T$$
$$P = [1, 1, 1, 2, 2, 2, 3, 4]^T, \quad N_p = 28, \quad N_c = 13$$
$$1.01 \times 10^7 \leq |\mathcal{G}_3| \leq 2.13 \times 10^{14}$$

The NSCs for this case study are now listed with expanded details on reduced potential adjacency matrix A_R. A few of the constraints utilize insights from the physical modeling of the suspension architectures.

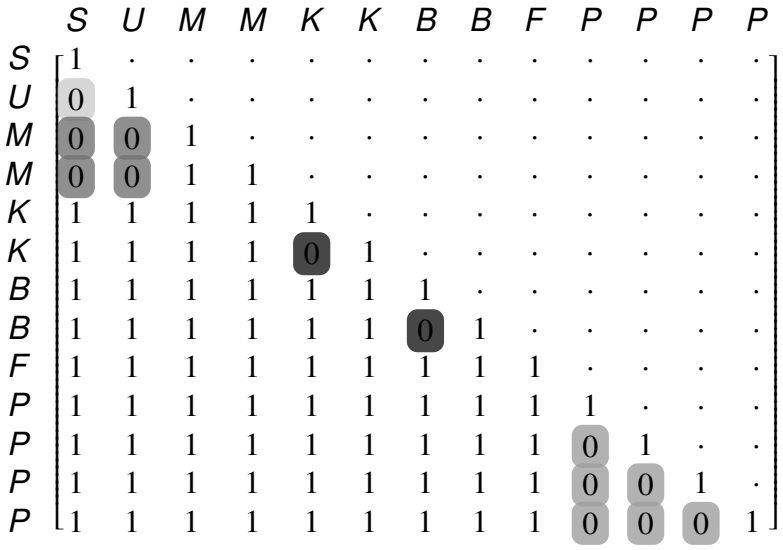
- S_1, S_3 and S_4 with M = [1 1 0 0 0 0 0 0] enforcing that both the sprung and unsprung masses must be connected and all components not connected to these two are removed.
- S_6 with P as the number of unique edges.
- $A_R(2,1) = 0$ for S_7 to avoid a direct connection between the sprung and unsprung masses as it would defeat the purpose of a suspension to isolate the masses. If this constraint was not added, 1/27 of graphs generated from a pure PM approach would contain this connection.
- $A_R(3,1) = A_R(3,2) = 0$ for S_7 since a feasible graph cannot have either S or U be connected to M as there would not be a path between between S and U. Therefore, no unique graphs are lost with this constraint. Rather, a more efficient enumeration results.
- $A_R(8,8) = A_R(7,7) = A_R(8,7) = 0$ for S_7 so no parallel components can be connected to each other. This greatly reduces the number of graphs generated by providing some specific structure on the number and type of parallel connections in the architectures.
- $A_R(4,4) = A_R(5,5) = 0$ for S_7 so no two K or B components can be connected in series since there are straightforward relationships to combine these series elements into a single equivalent component. By eliminating this type of connection when generating graphs, we have a substantially smaller number of graphs to evaluate.
- No parallel connection path can exist between a connected S or U as these masses would not be isolated. This is slightly different than the NSCs in Sec. 3.1 and is checked after S_7.
- Cycles appear with single parallel components where the components in the cycle would not appear in the dynamic model based on the properties of a parallel connection (e.g., with the cycle $P \overset{K}{\leftrightarrows} B$, neither K nor B would appear in the dynamic model). Graphs with these cycles are declared unuseful since an isomorphic graph with the parallel cycle removed would already appear in the set of graphs generated by the tree algorithm.
- Many series connections between the 2-port components are interchangeable (e.g., $F - K$ and $K - F$ in series are physically equivalent). Therefore, these interchangeable series components are combined into a single equivalent component type (modifying the graph) and checked for isomorphisms. For the previously mentioned example, the equivalent component type would be FK based on alphabetical ordering of the original component labels.

The complete A_R is shown in Fig. 14a and its expansion to the potential adjacency matrix in Fig. 14b noting that both of these matrices are symmetric. Figure 14b has 1s on the diagonal since self-connections should be allowed so the desired graph structure space is covered. For example, we might want to consider graphs where all components are present except a single K component and this only can occur if the detached K is connected to itself (and later removed).

The results are presented in similar manner to the previous case studies: 4.7×10^{21} adjacency

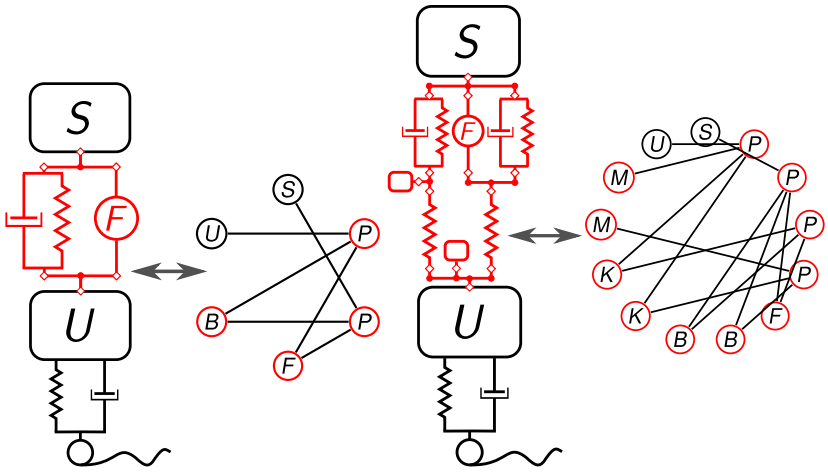


(a) Reduced potential adjacency matrix A_R.



(b) Potential adjacency matrix.

FIGURE 14: Suspension case study matrices for S_7 and the tree search algorithm.



(a) *PM* 49073434464833. **(b)** *PM* 48728499790028.

FIGURE 15: Two architectures for the suspension case study.

matrices[4] ($\mathcal{G}_0$); 2.1×10^{14} *PM*s ($\mathcal{G}_1$); 1.6×10^8 candidate graphs generated through the basic tree search algorithm; 3.2×10^7 remaining candidate graphs after initial port-type isomorphism filter; 1.9×10^6 feasible graphs ($\mathcal{G}_2$); and 13,727 unique graphs ($\mathcal{G}_3$). 2×10^9 graph comparisons were needed and 3×10^7 required a full isomorphism check. Two unique architectures are shown in Fig. 15 with Fig. 15a being a common architecture [38].

6 Discussion

It is clear in the case studies that the number of unique designs is much smaller than the upper bounds given by either permutations of the adjacency matrix or a *PM* approach. We also can directly visualize the effect of adding specific NSCs. NSCs limited the architecture design space, but in a predictable manner. One such example was not allowing parallel components to be connected together in Case Study 3. A 5-port parallel connection was no longer possible (i.e., a 3-port and 4-port parallel components connected), but this NSC greatly reduced the number of graphs generated excluding many infeasible graphs, such as ones where parallel connection paths existed between a connected S or U. Therefore, the addition of this NSC was a decision based on the tradeoff between coverage of the architecture design space and efficiency. Case Study 3 also demonstrated that fairly large problem sizes can be enumerated with the improved tree search algorithm provided enough constraints are present. Also, all possible subgraphs that are connected and complete appear in the set of unique designs without any NSCs (e.g., all graphs in Fig. 11 appear as subgraphs in Fig. 10).

All reported unique solutions have a corresponding *PM* number. This number may not be unique since other *PM* numbers maybe isomorphic to the resulting G^{CC}. We see an example of two different *PM*s producing isomorphic graphs (*PM* 462 in Fig. 9c and *PM* 678 in Fig. 10g).

[4]This represents the fairest comparison as the proper permutations of the ports representation of the potential adjacency matrix in Fig. 14b without the parallel components. There are 5.4×10^8 adjacency matrices with a component representation but this suffers from the unknown parallel connection issue discussed in Sec. 2. There are 3.0×10^{23} adjacency matrices directly using the matrix in Fig. 14b.

While checking for isomorphisms can be computationally demanding, there typically is only a small subset of graphs that need the full isomorphism check as many comparisons fail with the simple checks and filters. Algorithm 3 can be useful to any architecture design problem no matter how the set of colored graphs is generated. Many of the results and algorithms assumed simple components, but structured components (i.e., a planetary gear) can be readily included. Replacing a simple component type with an equivalent structured component type would simply have the effect of increasing the number of unique designs.

The previous sections only considered enumeration constructing a specific graph structure space. However, many problems are too large for the proposed enumeration algorithms because computational limits (e.g., memory needed to store the graphs and computation time of Alg. 3) and evaluation limits (i.e., too many graphs are generated and we cannot evaluate all of them). Therefore, we need to consider methods that provide suitable *exploration* of the desired design space.

Both the pure *PM* approach and the tree search algorithms have nice properties such as the high likelihood of producing feasible, nonisomorphic graphs while not limiting the design space. A stochastic sampling of the unique integers between 1 and $\mathcal{D}(N_p)$ can produce any arbitrary number of *PM* graphs. However, more structured sampling approaches may be preferred. Consider the unique graphs in Fig. 10. We could have tested all *PM*s between 227 to 913 and found all unique graphs. A *PM* approach does exhibit some interesting similarly-preserving properties (e.g., the graphs for a given *PM* number and the next integer value have a high likelihood of containing similar edges). Further exploration of the structural properties of *PM* graphs could lead to better sampling techniques that still cover the desired design space.

We can further consider ways to structure the exploration space with the coupon collector's problem. This problem, stated in a form relevant to this article, is:

> There are n unique graphs and at each trial, a new graph is chosen at random (with replacement). Let m be the number of trials. What is the expected number of trials such that all n unique graphs are selected?

The expected number of trials needed grows as $O(n\ln(n))$ [39]. Some of the assumptions in the problem are not directly satisfied such replacement and the probability distribution of choosing a particular unique graph but further study on the structure of $\mathcal{G}_1$ may yield exploration that 'collects' most of the unique graphs in a more efficient manner.

The tree search algorithms may also be used for exploration. On Line 5 of Alg. 3, we can randomly select an edge to add from I instead of trying all possible edges. Therefore, the tree can be explored stochastically. Since the number of branches from a leaf varies, the probability of arriving at a certain final edge set is not equal (these probabilities can be calculated by assuming the tree is a Markov chain). Since the tree search algorithms cover the same desired graph structure space as the pure *PM* approach, can we selectively sample the tree and have some predictions on when all unique designs are found? These questions are left as future work items.

Finally, it is important to describe the specific uses of the proposed algorithms. They are suitable for problems that are represented by undirected colored graphs under the component/port paradigm [2, 11, 19]. Enumeration is appropriate for certain problem classes (primarily determined by size). It may also be appropriate for searching for all possible enhancing structures [11]. Enumeration has been useful for generating lists of organic molecules [34, 35, 40], finding all geometries of electrical circuits [41], identifying all biological network architectures that achieve specific

behaviors [42], enumerating different gear trains [43, 44], and determining all hybrid powertrain configurations for a set list of components [24].

Exploration is suitable for sampling the design space for much larger problems [3]. These samples could be used as visualizations for expert evaluation or starting points for generative approaches. The unrestricted graphs from a *PM* approach could also be used in conjunction with feature extraction algorithms to develop generative rules that are not based solely on experience and intuition (where the features are subgraphs that provide desired benefits to the architecture) [45].

7 Conclusion

Architecture design is a challenging problem and this article presents some theory for candidate architectures with perfect matchings. A *PM* approach is a graph numerical representation scheme that completely covers the design space that is needed in many architecture design problems. It ensures certain frequency and degree requirements are met on specific list of potential components. Enumeration of architecture design spaces can provide coverage and insights not currently possible with generative design approaches since enumeration approaches allow the designer to make specifications that they understand such as constraints and potential components, rather than rules for how things should be connected.

A number of general network structure constraints are fully outlined with the specifics of checking their satisfaction with available graph analysis tools. The colored graph isomorphism problem is discussed in great detail including the distinction between port-type and component-type isomorphisms. The limited number of full isomorphism checks and the efficiency of Alg. 3 demonstrate that larger than expected architecture design spaces can be obtained. A basic and improved tree search algorithm that avoids port-type isomorphisms was shown and is a primary example of how constraints can be naturally satisfied without reducing the design space.

The various case studies are initial insights into the true nature of the class of architecture problems studied in this article. Consider again that there are only 12 unique graphs in Fig. 12 of 2,097,152 adjacency matrices and 135,135 *PM*s. For the suspension case study, a wide variety of network structure constraints were included based on natural requirements such as no direct connection between the sprung and unsprung masses. Other constraints were added to avoid duplicate dynamic models such as the avoidance of parallel cycles and series connections between the 2-port components. Moreover, constraints based on the experience and intuition of the designer were also included which limited the design space in a predictable manner such as the requirement that no parallel components can be connected together.

Future graph generation algorithms can use these insights to suitably address the unique challenges associated with architecture design problems. A number of directions are possible with the *PM* framework including deeper analysis of the structural properties of *PM* graphs, reduction of the number of graphs generated by the tree search algorithm, and the development of structured sampling approaches that result in nearly all unique graphs. In addition, there are ongoing studies to evaluate the set of suspension architectures generated for Case Study 3 based on their performance with respect to an architecture-dependent design optimization formulation (inner-loop problem of Prob. (1) based on the formulation in Ref. [38]).

Acknowledgments

The authors would like to thank Traci Spencer for her contributions. The opinions expressed here are solely those of the authors.

References

[1] Crawley, E., Weck, O. d., Eppinger, S., Magee, C., Moses, J., Seering, W., Schindall, J., Wallace, D., and Whitney, D., 2004. The Influence of Architecture in Engineering Systems. Engineering Systems Monograph, Massachusetts Institute of Technology, Mar.

[2] Mittal, S., and Frayman, F., 1989. "Towards a Generic Model of Configuration Tasks". In International Joint Conference on Artificial Intelligence, pp. 1395–1401.

[3] Wyatt, D. F., Wynn, D. C., and Clarkson, P. J., 2014. "A Scheme for Numerical Representation of Graph Structures in Engineering Design". *Journal of Mechanical Design,* ***136***(1), Jan., p. 011010. doi: 10.1115/1.4025961

[4] Cagan, J., Campbell, M. I., Finger, S., and Tomiyama, T., 2005. "A Framework for Computational Design Synthesis: Model and Applications". *Journal of Computing and Information Science in Engineering,* ***5***(3), Sept., pp. 171–181. doi: 10.1115/1.2013289

[5] 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". *Journal of Computing and Information Science in Engineering,* ***11***(2), June, p. 021003. doi: 10.1115/1.3593409

[6] Chan, J., Fu, K., Schunn, C., Cagan, J., Wood, K., and Kotovsky, K., 2011. "On the Benefits and Pitfalls of Analogies for Innovative Design: Ideation Performance Based on Analogical Distance, Commonness, and Modality of Examples". *Journal of Mechanical Design,* ***133***(8), Aug., p. 081004. doi: 10.1115/1.4004396

[7] Linsey, J. S., Tseng, I., Fu, K., Cagan, J., Wood, K. L., and Schunn, C., 2010. "A Study of Design Fixation, Its Mitigation and Perception in Engineering Design Faculty". *Journal of Mechanical Design,* ***132***(4), Apr., p. 041003. doi: 10.1115/1.4001110

[8] Deshmukh, A. P., Herber, D. R., and Allison, J. T., 2015. "Bridging the Gap between Open-Loop and Closed-Loop Control in Co-Design: A Framework for Complete Optimal Plant and Control Architecture Design". In 2015 American Control Conference, pp. 4916–4922. doi: 10.1109/ACC.2015.7172104

[9] Hooshmand, A., Campbell, M. I., and Shea, K., 2012. "Steps in Transforming Shapes Generated with Generative Design into Simulation Models". In International Design Engineering Technical Conferences, no. DETC2012-71056, pp. 883–892. doi: 10.1115/DETC2012-71056

[10] Khetan, A., Lohan, D. J., and Allison, J. T., 2015. "Managing Variable-Dimension Structural Optimization Problems Using Generative Algorithms". *Structural and Multidisciplinary Optimization,* ***52***(4), Oct., pp. 695–715. doi: 10.1007/s00158-015-1262-8

[11] Münzer, C., Helms, B., and Shea, K., 2013. "Automatically Transforming Object-Oriented Graph-Based Representations Into Boolean Satisfiability Problems for Computational Design Synthesis". *Journal of Mechanical Design,* ***135***(10), Oct., p. 101001. doi: 10.1115/1.4024850

[12] Guo, T., 2014. "Design of Genetic Regulatory Networks". M.S. Thesis, University of Illinois at Urbana-Champaign, Urbana, IL, USA, May.
[13] Schmidt, L. C., and Cagan, J., 1997. "GGREADA: A Graph Grammar-based Machine Design Algorithm". *Research in Engineering Design,* ***9***(4), Dec., pp. 195–213. doi: 10.1007/BF01589682
[14] Schmidt, L. C., Shetty, H., and Chase, S. C., 2000. "A Graph Grammar Approach for Structure Synthesis of Mechanisms". *Journal of Mechanical Design,* ***122***(4), Dec., pp. 371–376. doi: 10.1115/1.1315299
[15] Hornby, G. S., Lipson, H., and Pollack, J. B., 2003. "Generative Representations for the Automated Design of Modular Physical Robots". *IEEE Transactions on Robotics and Automation,* ***19***(4), Aug., pp. 703–719. doi: 10.1109/TRA.2003.814502
[16] Bryant, C. R., McAdams, D. A., Stone, R. B., Kurtoglu, T., and Campbell, M. I., 2005. "A Computational Technique for Concept Generation". In ASME International Design Engineering Technical Conferences, no. DETC2005-85323, pp. 267–276. doi: 10.1115/DETC2005-85323
[17] Starling, A. C., and Shea, K., 2005. "A Parallel Grammar for Simulation-Driven Mechanical Design Synthesis". In ASME International Design Engineering Technical Conferences, no. DETC2005-85414, pp. 427–436. doi: 10.1115/DETC2005-85414
[18] 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". *Research in Engineering Design,* ***23***(1), Jan., pp. 17–52. doi: 10.1007/s00163-011-0112-y
[19] Snavely, G. L., and Papalambros, P. Y., 1993. "Abstraction as a Configuration Design Methodology". In Advances in Design Automation, Vol. 65, pp. 297–305.
[20] Godsil, C., and Royle, G., 2001. *Algebraic Graph Theory*. Springer. doi: 10.1007/978-1-4613-0163-9
[21] Diestel, R., 2000. *Graph Theory*, 4th ed. Springer.
[22] Rispoli, F. J., 2007. *Applications of Discrete Mathematics*, updated ed. McGraw-Hill, ch. Applications of Subgraph Enumeration, pp. 241–262.
[23] Wu, Z., Campbell, M. I., and Fernández, B. R., 2008. "Bond Graph Based Automated Modeling for Computer-Aided Design of Dynamic Systems". *Journal of Mechanical Design,* ***130***(4), Apr., p. 041102. doi: 10.1115/1.2885180
[24] Bayrak, A. E., Ren, Y., and Papalambros, P. Y., 2016. "Topology Generation for Hybrid Electric Vehicle Architecture Design". *Journal of Mechanical Design,* ***138***(8), Aug. doi: 10.1115/1.4033656
[25] Sloane, N. J. A. Sequence A001147. The On-Line Encyclopedia of Integer Sequences.
[26] Herber, D. R., 2015. Perfect Matchings of a Complete Graph. Online. http://www.mathworks.com/matlabcentral/fileexchange/52301.
[27] Lawler, E., 1976. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston.
[28] McKay, B. D., and Piperno, A., 2014. "Practical Graph Isomorphism, II". *Journal of Symbolic Computation,* ***60***, Jan., pp. 94–112. doi: 10.1016/j.jsc.2013.09.003
[29] Babai, L., 2015. Graph Isomorphism in Quasipolynomial Time. Online, Dec. arXiv:1512.03547.
[30] Python-igraph manual. Online. http://igraph.org/.
[31] Cordella, L. P., Foggia, P., Sansone, C., and Vento, M., 2001. "An Improved Algorithm

for Matching Large Graphs". In IAPR TC-15 Workshop on Graph-based Representations in Pattern Recognition, pp. 149–159.

[32] Königseder, C., and Shea, K., 2016. "Comparing Strategies for Topologic and Parametric Rule Application in Automated Computational Design Synthesis". *Journal of Mechanical Design,* ***138***(1), Jan., p. 011102. doi: 10.1115/1.4031714

[33] Read, R. C., 1978. "Every One a Winner or How to Avoid Isomorphism Search When Cataloguing Combinatorial Configurations". *Annals of Discrete Mathematics,* ***2***, pp. 107–120. doi: 10.1016/S0167-5060(08)70325-X

[34] Faulon, J.-L., Churchwell, C. J., and Visco Jr, D. P., 2003. "The Signature Molecular Descriptor. 2. Enumerating Molecules from Their Extended Valence Sequences". *Journal of Chemical Information and Modeling,* ***43***(3), pp. 721–734. doi: 10.1021/ci020346o

[35] Carhart, R. E., Smith, D. H., Brown, H., and Djerassi, C., 1975. "Applications of Artificial Intelligence for Chemical Inference. XVII. An Approach to Computer-Assisted Elucidation of Molecular Structure". *Journal of the American Chemical Society,* ***97***(20), Oct., pp. 5755–5762. doi: 10.1021/ja00853a021

[36] Colbourn, C. J., and Read, R. C., 1979. "Orderly Algorithms for Generating Restricted Classes of Graphs". *Journal of Graph Theory,* ***3***(2), pp. 187–195. doi: 10.1002/jgt.3190030210

[37] Herber, D. R., Guo, T., and Allison, J. T., 2016. PM Architectures Project. Online. https://github.com/danielrherber/pm-architectures-project.

[38] Allison, J. T., Guo, T., and Han, Z., 2014. "Co-Design of an Active Suspension Using Simultaneous Dynamic Optimization". *Journal of Mechanical Design,* ***136***(8), Aug., p. 081003. doi: 10.1115/1.4027335

[39] Flajolet, P., Gardy, D., and Thimonier, L., 1992. "Birthday Paradox, Coupon Collectors, Caching Algorithms and Self-Organizing Search". *Discrete Applied Mathematics,* ***39***(3), Nov., pp. 207–229. doi: 10.1016/0166-218X(92)90177-C

[40] Ruddigkeit, L., Deursen, R. v., Blum, L. C., and Reymond, J.-L., 2012. "Enumeration of 166 Billion Organic Small Molecules in the Chemical Universe Database GDB-17". *Journal of Chemical Information and Modeling,* ***52***(11), pp. 2864–2875. doi: 10.1021/ci300415d

[41] Foster, R. M., 1932. "Geometrical Circuits of Electrical Networks". *Electrical Engineering,* ***51***(1), Jan., pp. 309–317. doi: 10.1109/EE.1932.6429606

[42] Ma, W., Trusina, A., El-Samad, H., Lim, W., 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

[43] Pennestrì, E., and Valentini, P. P., 2015. "Kinematics and Enumeration of Combined Harmonic Drive Gearing". *Journal of Mechanical Design,* ***137***(12), Oct., p. 122303. doi: 10.1115/1.4031590

[44] Castillo, J. M. d., 2002. "Enumeration of 1-DOF Planetary Gear Train Graphs Based on Functional Constraints". *Journal of Mechanical Design,* ***124***(4), Dec., p. 723. doi: 10.1115/1.1514663

[45] Berlingerio, M., Bonchi, F., Bringmann, B., and Gionis, A., 2009. *Machine Learning and Knowledge Discovery in Databases*, Vol. 5781. Springer, ch. Mining Graph Evolution Rules, pp. 115–130. doi: 10.1007/978-3-642-04180-8_25

A Definitions of Selected Acronyms and Symbols

A_R	reduced potential adjacency matrix
A_C	connectivity matrix
C	colored label set representing distinct component types
$\mathcal{D}$	double factorial function
E	edges of a graph
G	graph
G^P	ports graph
G^I	interconnectivity graph
G^{CC}	connected components graph
G^{CP}	connected ports graph
$\mathcal{G}$	graph structure space
$\mathcal{G}_1$	all graphs that satisfy (C, R, P) and have every port filled
$\mathcal{G}_2$	all graphs in $\mathcal{G}_1$ that satisfy all NSCs
$\mathcal{G}_3$	all graphs in $\mathcal{G}_2$ that are unique
K_n	complete graph with n vertices
L	labels of a colored graph
M	boolean vector indicating the mandatory components
NSC	network structure constraint
N_c	maximum number of components in specified (C, R, P) problem
N_p	maximum number of ports in specified (C, R, P) problem
n_p	number of ports in a candidate architecture
P	vector indicating the number of ports for each component
PM	perfect matching
$\mathcal{P}(n, N)$	edge set for the nth PM of K_N
R	vector indicting the number of replicates for each component
S_1	a specific NSC, every graph must be a connected graph
S_2	a specific NSC, every graph can only have a maximum number of a given component type
S_3	a specific NSC, every graph must have a specific number of certain component types
S_4	a specific NSC, every graph must have specific component types connected to each other
S_5	a specific NSC, every graph must have vertices whose number of unique edges is within a specific range
S_6	a specific NSC, every graph must have vertices with a specified number of unique connections
S_7	a specific NSC, every graph must have edges between vertices that are feasible
$\mathcal{T}$	number of permutations of the undirected graph's adjacency matrix
V	vertices of a graph