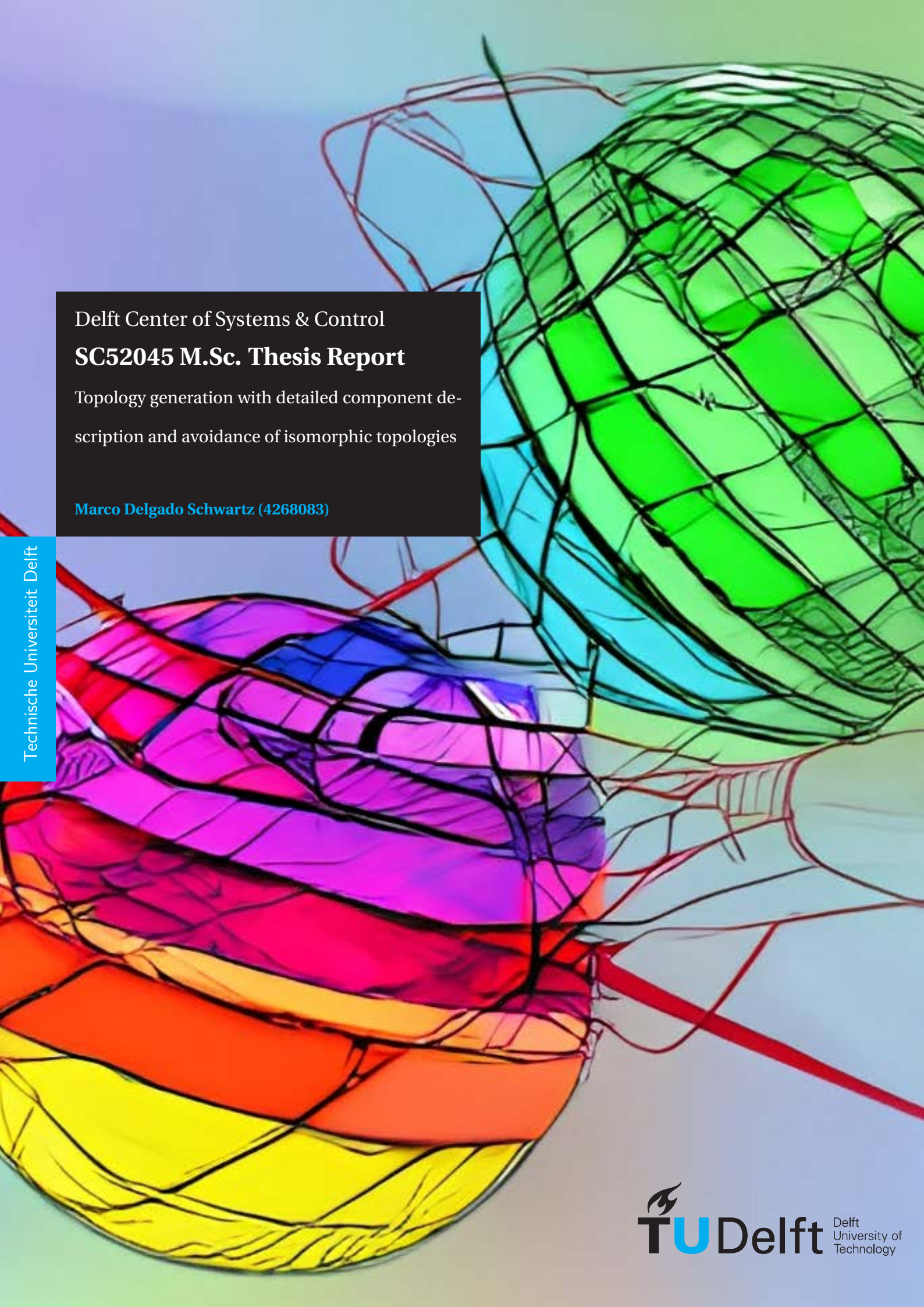


Delft Center of Systems & Control

SC52045 M.Sc. Thesis Report

Topology generation with detailed component description and avoidance of isomorphic topologies

Marco Delgado Schwartz (4268083)

Technische Universiteit Delft

TU Delft Delft University of Technology

SC52045 M.Sc. Thesis Report

Topology generation with detailed component description and avoidance of isomorphic topologies

by

Marco Delgado Schwartz (4268083)

MSc thesis main supervisor:	Dr. Peyman Mohajerin Esfahani
Literature review daily supervisor:	Dr. Shengling Shi
Thesis daily supervisor:	Phd. Mohammad Boveiri
Company (TNO) supervisor:	Dr. Steven Wilkins

Delft Center of Systems & Control (DCSC)
Technical University of Delft
August 19, 2024
Delft, Netherlands

TU Delft Delft University of Technology

Cover image was generated using the Craiyon AI image generator (craiyon. com).

PREFACE

The European Union (EU) has set the goal to have a carbon-neutral economy by 2050 [1]. To achieve this, a key sector to focus on is the transportation sector [2, 3]. It will be especially challenging though to decarbonise the larger vehicles from the transportation sector, the trucks, ships and airplanes [3].

The typical way to respond to this or any other design challenge is by making use of the top-down/sequential design process, where first the larger parts of the design are established, and later the finer details are developed. However such an approach cannot guarantee that the design at the end is optimal or anywhere close to it [4, 5]. This makes the sequential design approach ill-suited to tackle the challenge of decarbonising the EU transportation sector. Co-Design on the other hand, a simultaneous design approach, can guarantee optimality [4, 5], and would be a much more appropriate approach if it wasn't for it being limited to only consider around 10-20 components [6, 7, 8, 5].

The work in this report first investigated where this limitation of 10-20 components comes from. It is described how the problem's source does not originate from topology optimisation, where most research has gone into, but rather that it originates from the topology generation process. The suspicion was that the vast number of isomorphic topologies that are created during the topology generation process make the topology generation process intractable with the current methods when considering more than 10-20 components. Adding more data to describe the components in more detail or adding multiple instances for each component that can be used multiple times across a design only increases the number of isomorphic topologies even further, and with it the computational cost to complete the topology generation process. Since the results of the topology generation process are fed to the topology optimisation process, the entire Co-Design process is then influenced by these isomorphic topologies.

The work that follows therefore focused on topology generation for Co-Design by testing a candidate method that allows topologies to be generated without any of the isomorphic topologies while describing components in more detail than what has been done in literature. The approach taken here relied, unlike what has been done so far, on the use of adjacency matrices and the interconnected system model. With this, new insight into topology generation for Co-Design was developed. The use of adjacency matrices were also key to automate the formation of constraints, which allowed to generate topologies for any set of components, also a first in the literature. The program TopoGen was developed to make this possible. The results that were obtained showed how using requirement chains and ordering instances, at least in simple cases, the 10-20 components limitation is fully overcome. This confirmed that the presence of isomorphic topologies the cause was for 10-20 components limitation in Co-Design. Still, for more complex cases, mistakes were present. However it was also shown how these too may be prevented in the future. This means that with further research and development, the candidate method presented here can be adjusted so that Co-Design can be applied even to complex cases, and therefore be used to develop solutions needed to decarbonise the EU transportation sector.

The report is structured as follows: After the introduction in chapter 1, in chapter 2 the current methods around Co-Design are reviewed. It is revealed that the sub-process of topology generation the reason is as to why Co-Design is restricted to such small sets. This leads to defining the research questions, which are presented in section 2.3. The research questions define the research scope of this thesis, and are all focused on the topic of topology generation for Co-Design. The sections that follow, in chapter 3, present the work that has been done to answer these questions, as well as the results that were obtained along the way. In section 3.1, the basic setup that was used is presented. The sections that follow, from section 3.2 to section 3.4, build on this setup and present intermediate results. In section 3.5 then the candidate method to generate topologies without ismorphic topologies is presented and the test results are shown. The report is finally wrapped up in chapter 4 by summarising the contributions that have been made towards the research questions, as well as presenting a list of recommendations on how to continue the research.

I would like to thank all of my supervisors for the patience and support they have given me throughout my thesis work. From the beginning Peyman has been vocal and proactive in helping me find a topic I was passionate to work on. Shengling guided me through the literature review, and helped me condense my ideas into useful research building blocks. These eventually made me come across the work done by TNO in the field of Co-Design. Upon contacting them on their work, and presenting them what I had been working on so far and planning to do next, Steven invited me to complete my graduation work at their research department. Since then he has been helping me with useful reviews of my work and progress, exposed me to relevant research and ideas, and ensured I could complete my research work at TNO despite it taking longer than expected. On the university side, Mohammad replaced Shengling after Shengling had left for his Post-doc position at MIT. Mohammad has been a reliable contact person throughout my research project. I was able to approach him at all times and he assisted me through the questions I had or the difficulties I had encountered.

Thanks to the freedom and support I was given throughout my entire thesis, I was able to develop and research a topic I was passionate about. While I may have achieved now enough results to finish, and seem to have even made some noteworthy contributions to the field of Co-Design, I by no means feel like I am done. I feel like I am just getting started. What better feeling is there to finish one's engineering MSc?

NOMENCLATURE

Abbreviations

CLD	Component Level Description
CPG	Constrained Permutation Generator
CSP	Constrained Satisfaction Problem
EU	European Union
FRD	Functionalities-Requirements-Diagram
IEA	International Energy Agency
ISM	Interconnected System Model
MRQ	Main Research Question
PLD	Port Level Description
PPLD	Property-Port Level Description
SAT	Satisfiability Problem
SRQ-i	Supplementary Research Question i
TF	Feasible topologies
TP	Potential topologies
TPB	Potential topologies with bijective connections
TPBC	Potential topologies with bijective compatible connections
UAL	Unweighted Adjacency List
UAM	Unweighted Adjacency Matrix

Operators

$(f, \geq)$	Upper bound of the function f
$(f, \leq)$	Lower bound of the function f

$[\cdot]$ List operator, such as the list of property ULIDs [P-ID]

$[\text{P-ID}]_{[\text{C-ID,V-ID}]}$ List of property ULIDs, sorted first by the order of the component ULIDs, and then the unit type ULIDs.

$[a..b]$ List of integers from a to b

$\circ$ Fuzzy/Boolean composition operator

$\ell(v)$ Length operator, which provides the number of elements in v.

$\ell_n(M)$ Length operator, which provides the number of elements in axis n of the matrix M. (0 for rows, 1 for columns).

$\wedge/.$ Logical and operator

$\langle\cdot\rangle$ Type operator, such as the node type $\langle N\rangle$

$\vee$ Logical or operator

$\mu([L_1]\times[L_2])$ Membership operator, that defines which elements in L_1 and L_2 belong together.

$\mu_r([L_1]\times[L_2])$ Reduced membership operator, that defines which elements in L_1 and L_2 belong together, but removing all rows and columns that have no connections.

$\nu([L_1]\times[L_2])$ Potential connections operator, that defines which elements in L_1 and L_2 might belong together.

$\times$ Cartesian cross product

$\zeta([L_1]\times[L_2])$ Compatible connection operator, that defines which elements in L_1 and L_2 are compatible with each other.

Variables

(V_i, V_j) Edge between node i and node j

[C-ID] List of component ULIDs

[D-ID] List of domain type ULIDs

[H-ID] List of instance ULIDs

[N-ID] List of node type ULIDs

[P-ID] List of property ULIDs

[R-ID] List of port ULIDs

[T-ID] List of component type ULIDs

[V-ID] List of unit type ULIDs

$\mathbb{B}$ Boolean number set

$\mathbb{R}$ Real number set

$\mathbb{Z}$ Integer number set

$\mathcal{C}$ Set of constraints for CSP

$\mathcal{D}$ Set of domains for CSP

$\mathcal{S}_c$ Set of components

$\mathcal{S}_r$ Set of requirements

$\mathcal{T}_f$ Set of feasible topologies such that $\mathcal{T}_f \subseteq \mathcal{T}_{p,bc} \subseteq \mathcal{T}_{p,b} \subseteq \mathcal{T}_p$

$\mathcal{T}_p$ Set of potential topologies such that $\mathcal{T}_p := \{M_0..M_{\ell_0(M)\ell_1(M)}\}$

$\mathcal{T}_{f,q}$ Number of branches in requirement chain, used to count feasible topologies from requirement chain.

$\mathcal{T}_{f,r}$ Number of components that meet output requirement, used to count feasible topologies from requirement chain.

$\mathcal{T}_{p,bc}$ Set of potential topologies with bijective compatible connections such that $\mathcal{T}_{p,bc} \subseteq \mathcal{T}_{p,b} \subseteq \mathcal{T}_p$

$\mathcal{T}_{p,b}$ Set of potential topologies with bijective connections such that $\mathcal{T}_{p,b} \subseteq \mathcal{T}_p$

$\mathcal{X}$ Set of variables for CSP

$\mathcal{X}_e$ Set of edge variables

$\mathcal{X}_n$ Set of node variables

$\boldsymbol{u}$ Input vector

$\boldsymbol{y}$ Output vector

L Local map UAM

L_n Local map UAM of component n

M Feedback gain of ISM/Connection UAM

Q The UAM which specifies the connections from input ports to output ports.

r Number of connections placed between available ports

R_u	Number of input ports
R_y	Number of output ports
u	Input from $\boldsymbol{u}$; Input set from node type.
V_i	Node i
y	Output from $\boldsymbol{y}$; Output set from node type.

Contents

1

INTRODUCTION

The European Union (EU) has set the goal to have a carbon-neutral economy by 2050 [1]. In 2020 the transportation sector was responsible for about one-fifth of the global carbon dioxide emissions, with 45% of the emissions coming from cars, 30% from trucks, 12% from airplanes, and 11% from ships [9]. However, as is explained in [2], between 1990 and 2018, while all other sectors have been able to reduce their emissions, the transportation sector has instead been increasing them. Moreover, projections show that from 2019 to 2070, global transport (measured in passenger kilometres) will double, the number of passenger and freight airplanes will triple, and car ownership will increase by 60% [3]. Emissions from shipping alone could increase up to 130% by 2050 from 2008 levels [10]. In order to meet the goal of carbon neutrality by 2050 set by the EU, the transportation sector is therefore a key sector to focus on.

However, this comes with several complications. For one, while the International Energy Agency (IEA) acknowledges the rapid progress that is being made in reducing the emissions from cars, it also states that it will be significantly harder to do this with trucks, ships and airplanes due to their increased size and complexity [11, 3]. Additionally the service life of ships and airplanes are around 30 years, and the current fleet is only halfway through this life [12, 3]. That means that only around 2040, if alternatives exist, these ships and airplanes will start to be replaced. Indeed, the IEA points at ships and airplanes to be the last ones to reduce their emissions [3]. However, even before 2040, and assuming the technology is mature enough, it also takes roughly five to ten years to design a new ship or airplane [13, 14, 15, 16] in the first place. Additionally, it is not just that ships or airplanes need to be redesigned, but also that the technologies that will make it possible to reduce or eliminate their emissions will also require major changes in the supply chain and the infrastructure that are needed for their construction, maintenance and operation [3]. All of this leads to the IEA describing that "reducing CO2 emissions in the transport sector over the next half-century will be a formidable task" [3]. Solutions need to be developed now so that this transition becomes successful.

> *Reducing CO2 emissions in the transport sector over the next half-century will be a formidable task. It will require structural shifts in the modes used to move people and freight, a shift to low-carbon forms of energy and a stronger focus on using energy more efficiently. This calls for a broad mix of technologies, many of which are at the early stages of development and commercialisation. These include vehicle, powertrain and*

> *engine technologies, and the infrastructure needed to support alternative fuels, as well as digital technologies and software to enable service providers to harness the power of data.* (Energy Technology Perspective 2020, IEA. [3])

The usual approach to develop these solutions, designs that meet some set of requirements, is a top-down/sequential design approach. This involves first figuring out the big parts of the design, and then proceeding to figuring out the ever finer details of it. Using this approach it has also been possible to design the trucks, ships and airplanes that operate today.

Nevertheless, as literature explains [4, 5], such a sequential design process comes with several issues. The issues arise from the fact that the first elements that were designed and put into place have been designed without considering the influence of the design elements that are considered in the later stages of the design process. The result is, no matter how rigorous this sequential design process is done, as [4] proceeds to show, there is zero guarantee that the design at the end will be optimal, or anywhere close to being optimal. Additionally, with the increased complexity that comes with bigger vehicles such as airplanes and ships, only few ways of how to assemble the components, the design architecture, can be considered [11]. However, for the energy transition, where new technologies such as batteries or hydrogen fuel cells need to be incorporated [3], multiple design architectures need to be considered. If this however is attempted, with the sequential design approach, again, no guarantee exists that developing these alternatives will be successful or close to performing as well as might be possible or necessary.

Fortunately, an alternative to the sequential design approach exists. This is the simultaneous design approach, more commonly referred to as Co-Design. This approach, unlike the sequential approach, can guarantee that a design ends up being optimal to the requirements it needs to satisfy [5, 4]. While the sequential design approach has the advantage that tasks can be divided and assigned to different people/departments, Co-Design instead is completed as one large optimisation problem with the end result being a tool that can automatically evaluate all possible design architectures, and retrieve the optimal ones.

Such a tool would be extremely helpful to design the solutions required for the energy transition set forward by the EU [17, 5]. Using Co-Design, design architectures that include new technologies can be considered, thereby helping to integrate these alternatives faster into working designs. But even in the case where alternative technologies are not considered, Co-Design can still explore design architectures that have not been examined before, and further optimise existing designs, always with the guarantee of optimality.

Unfortunately though, Co-Design does suffer from one great limitation: It only works when considering at most between 10-20 components for ones design [6, 7, 8, 5]. Beyond that the computational burden of solving the single optimisation problem is so high, it quickly becomes a discussion of having to wait decades or even centuries for one to obtain the optimal design [6].

Still, noteworthy achievements have been accomplished with Co-Design by bypassing the limitation of only being able to consider 10-20 components. For instance, in [5] it is summarised how research has gone into redesigning existing hybrid electric vehicles (HEVs) using the Co-Design approach instead. The redesigned HEVs had significant improvements in performance and reduced emissions. For example, in [18], cited by [5], after redesigning the HEVs considered in the paper, a remarkable 33% decrease in CO2 emissions have been found.

Two strategies are commonly employed to achieve these results despite the 10-20 components

limitation: The first one involves limiting the design architectures to a smaller selection [5], thereby reducing the number of designs to evaluate in the optimisation process behind Co-Design. The second strategy involves combining components together to form fewer but larger components [11], and therefore able to stay under the 10-20 component limitation. Both of these strategies however, due to rejecting in one form or another many of the potential design architectures, sacrifice the most important feature of Co-Design, namely that of guaranteeing the optimality of the final design. Additionally, the sacrifice increases when considering ever-larger vehicles.

Otherwise, the rest of the research effort, as is described in [5], has gone into testing different optimisation algorithms that can handle the highly nonlinear optimisation problem of finding the optimal design architecture from a list of architectures, as well as seeing how these calculations can be sped up. Nevertheless, these do not address the source of the problem, the 10-20 component limitation, as will be explained in chapter 2. The problem namely lies in generating all the different design architectures, i.e. the topology generation.

The research work presented here focuses on this limitation. The hope is that by better understanding how this limitation arises, an alternative approach to topology generation can be assembled, so that Co-Design can executed with plenty more components than just the 10 or 20.

To that end, in chapter 2 first the current methods with respect to Co-Design are described, while introducing important definitions. This is then used to explain from where the 10-20 components limitation comes. All of this put together is used to substantiate the research questions in section 2.3 that are the focus of the research work done here. In chapter 3 the method that was applied to answer these very research questions is presented, showing alongside the results that have been obtained along the way. The section is divided into five parts, where section 3.1 presents the basic setup that was used. The sections that follow, from section 3.2 to section 3.4, build on that setup and present intermediate results. In section 3.5 the candidate method to overcome the 10-20 components limitation is presented, as well as results that have been obtained with it. Finally, in chapter 4, the report is wrapped up by summarising the contributions that have been made towards the research questions. The section ends by providing a list of recommendations on how to continue with the research.

2

CURRENT METHODS & RESEARCH QUESTION

In the following section, starting with section 2.1, the current methods that are applied for Co-Design are briefly summarised, in context to what will be relevant for the remaining sections hereafter. (For a full review of the current methods that are applied for Co-Design, please refer to [5].) This is then followed by section 2.2, where the most significant limitations of the current methods will be presented. Having described the current methods and limitations, in the final section, section 2.3, the research questions for the thesis are presented.

2.1. CURRENT METHODS

Co-Design is initialised on a set of components $\mathcal{S}_c$ and a set of requirements $\mathcal{S}_r$, and the goal is to find what components and connections between components should be selected in order to deliver as an output an optimal design that meets the set of requirements. Because components and connections are selected to produce any design, the designs are actually referred to as topologies. In Figure 2.2 a flowchart of the Co-Design process is presented, where the blue parallelograms indicate the data that is being considered, and the two processes that it involves: topology generation and topology optimisation.

Consider Figure 2.1 as an example for a set of requirements, currently not specified any further, and a set of components with three components, a battery, a wheel and an electric motor. Each component has been given a unique label, V_1, V_2 and V_3. Finally, also the connections one might consider between the components, here represented only as undirected edges, are labelled as tuples of the two components they connect. For example the connection between V_2 (the battery) and V_1 (the electric motor) is labelled as (V_1, V_2). (Because here only undirected edges are considered, the equality $(V_1, V_2) = (V_2, V_1)$ holds.) Once these two sets have been established, topology generation, the first process in Co-Design, is executed, as also shown in Figure 2.2.

The goal of topology generation is to find out which of all the different connections that can be placed, should be chosen, in order to obtain a topology that can fulfil the set of requirements. As is already recognised in [8, 5], this results in a Constrained Satisfaction Problem (CSP), which is defined by a triplet $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ such that:

- $\mathcal{X} = \{X_1..X_n\}$ is the set of variables for the problem.
- $\mathcal{D} = \{\mathcal{D}_1..\mathcal{D}_n\}$ is the set of domains for each variables such that $X_i \in \mathcal{D}_i \ \forall i \in [1..n]$.

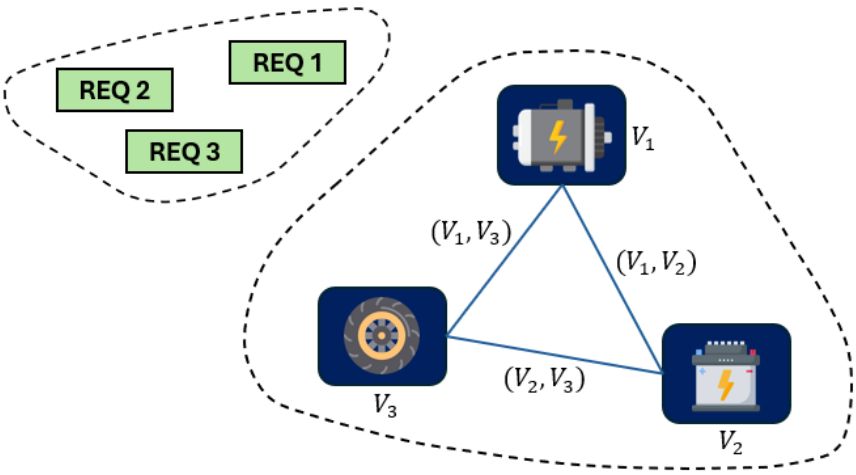


Figure 2.1: Example of a set of components $\mathcal{S}_c$ and a set of requirements $\mathcal{S}_r$.

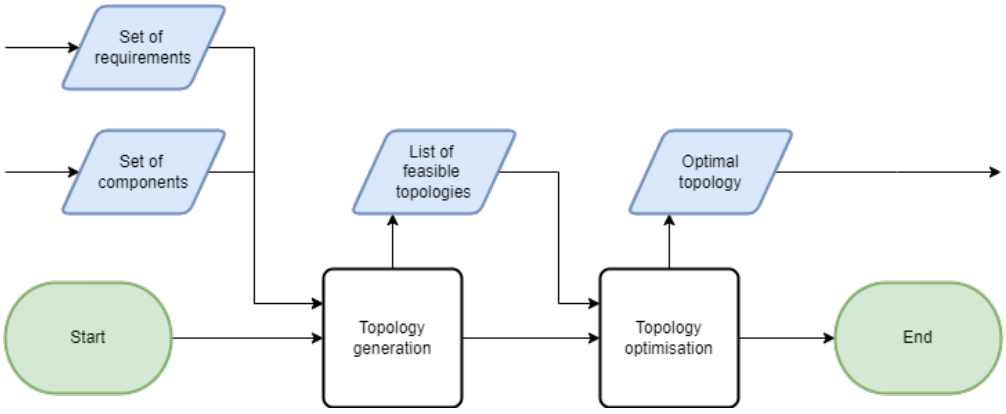


Figure 2.2: Flowchart of the process behind Co-Design.

- $\mathcal{C}$ is the set of constraints such that $\mathcal{C}_{ijk...} \subseteq \mathcal{D}_i \times \mathcal{D}_j \times \mathcal{D}_k \times ...$, representing a subset of the possible combinations of values of X_i,X_j,X_k...

Considering the example in Figure 2.1 and following the same strategy as in [8], the variables in $\mathcal{X}$ are each component and connection, i.e. $\mathcal{X} = \{V_1, V_2, V_3, (V_1, V_2), (V_2, V_3), (V_1, V_3)\}$. The set of variables $\mathcal{X}$ can thus be divided into two disjoint sets such that $\mathcal{X} = \{\mathcal{X}_n, \mathcal{X}_e\} : \mathcal{X}_n \cap \mathcal{X}_e = \emptyset$, with $\mathcal{X}_n = \{V_1, V_2, V_3\}$ representing the components and $\mathcal{X}_e = \{(V_1, V_2), (V_2, V_3), (V_1, V_3)\}$ the connections. The domain for all variables is $D_i = \mathbb{B} := \{0, 1\}$, i.e. components and connections can be turned on or off and can thus be represented as boolean variables. Finally, in [8], the constraints are defined. For example it is defined how many connections each component can make, where when referring back to the example shown in Figure 2.1, one might state that if the battery (an energy source) is selected, i.e. $V_2 = 1$, or the wheel (an energy sink), i.e. $V_3 = 1$, that both of them can only have one connection. The electric motor on the other hand, if selected, i.e. $V_1 = 1$, then it can have two connections. These constraints may be expressed like shown in (2.1):

$$\begin{aligned} V_2 &= (V_1, V_2) + (V_2, V_3) \\ V_3 &= (V_2, V_3) + (V_1, V_3) \\ 2V_1 &= (V_1, V_3) + (V_1, V_2) \end{aligned} \tag{2.1}$$

Additionally, from the example shown in Figure 2.1, one might state that the connection (V_2, V_3) never makes sense to place, since there is no compatible connection that can be placed between a battery and a wheel. In such a case a further constraint can be placed such that $(V_2, V_3) = 0$. Finally, the set of requirements is also translated as constraints that need to be met. Referring back to Figure 2.1, one might state as a requirement that one would like to have a wheel that spins. This might imply as a constraint that $V_3 = 1$.

With this approach [8] proceeds to define the rest of the constraints. The CSP is then solved using Prolog, which, as is explained in [8] is a popular solver to solve CSPs. Any solution that is provided by a CSP solver is referred to as a feasible topology (TF), where each TF specifies the values the variables in $\mathcal{X}$ take in order to meet all the specified constraints. In the example shown in Figure 2.1, one might obtain then as a TF that the battery needs to be connected to the electric motor, and the electric motor to the wheel. The result could then be presented as shown in Table 2.1:

X_i	V_1	V_2	V_3	(V_1, V_2)	(V_2, V_3)	(V_1, V_3)
$\mathcal{D}_i$	1	1	1	1	0	1

Table 2.1: TF for example shown in Figure 2.1.

As is shown in [8], the assembly of the constraints plays a crucial role. In their case, when using a set of 16 components, i.e. $|\mathcal{S}_c| = 16$, 16 variables are needed in $\mathcal{X}_n$ to specify the selection of the components, i.e. $|\mathcal{S}_c| = |\mathcal{X}_n|$. Additionally 136 variables in $\mathcal{X}_e$ are needed to specify any potential connection between the components. Since undirected edges are considered, the number of potential connections are $|\mathcal{X}_e| = \frac{1}{2}\left(|\mathcal{X}_n|^2 + |\mathcal{X}_n|\right)$. This leads to a total of 152 variables in $\mathcal{X}$, with the domain of each variable in $\mathbb{B}$. Then, without any constraints, the number of potential topologies (TPs) is as shown in (2.2), leading to a total number of 5.7×10^{45} TPs. This is an enormous number of topologies that amount to a total close to the number of atoms that exist on earth, which is estimated to be at 1.3×10^{50} [19]. (In fact it would take 18 components instead of 16 to surpass that number.) However, when solving the CSP, only $|\mathcal{T}_f| = 4'779$ TFs exist.

$$|\mathcal{T}_p| = 2^{\left(\frac{1}{2}|\mathcal{X}_n|^2 + \frac{3}{2}|\mathcal{X}_n|\right)} \tag{2.2}$$

The set of TFs $\mathcal{T}_f$ is then fed to the next process in Co-Design (see Figure 2.1), topology optimisation, where an optimisation over this list of TFs is executed in order to find the optimal topology, i.e. the topology that satisfies the set of requirements $\mathcal{S}_r$ at the lowest input cost. The topology optimisation also optimises each component parameter such as the size of the components, as well as developing optimal control for the topology being explored [5]. This leads to a highly nonlinear optimisation problem [4, 5]. As explained in [5], this is also the reason why genetic algorithms or particle swarm algorithms tend to be the algorithms used to complete the topology optimisation, two global search algorithms suitable for the optimisation problem considered the topology optimisation process.

2.2. LIMITATIONS

Having addressed in section 2.1 how the current state of the art deals with Co-Design, in this section, three problems are explained that limit Co-Design to work on only 10-20 components.

2.2.1. PROBLEM 1: REWRITING CONSTRAINTS

As was explained in section 2.1, the constraints play a crucial role in retrieving the feasible topologies (TFs) in the topology generation process. However, currently, there is no general method that can be applied to develop all of these constraints. Additionally it remains unknown what the impact is of each constraint, i.e. the number of potential topologies (TPs) the constraint eliminates as being not feasible. This leads to two big problems:

First, there is a huge risk that one or another constraint is forgotten. Considering that the number of TPs scales at an approximate rate of $2^{|\mathcal{X}_n|^2}$ (see (2.2)), and each of the topologies that are not eliminated by the constraints are deemed feasible, the number of TFs one might end up because of a forgotten constraint may make the next process, topology optimisation, prohibitively expensive or downright impossible to complete in full. As is explained in [5], in several studies, only a subset of 4-10 TFs are selected for the topology optimisation. The $4'779$ TFs found in [8] are already a challenge. Any constraint that is forgotten can quickly lead to the Co-Design process not being able to be completed at all.

Secondly, the current approach with the constraints forces that with any new set of components, the constraints need to be rewritten. This happens even if the new set of components differs by only one component compared to the original set of components. The consequence is that Co-Design, with the current approach, is an extremely arduous and risky process, as again one may forget a constraint or make a mistake.

Research is therefore needed to produce a reliable method for the constraints used in topology generation. Understanding the impact that each constraint has might help develop such a method.

2.2.2. PROBLEM 2: SLOW TOPOLOGY GENERATION

With the current Co-Design approach even with a small set of components, i.e. 10-20 components, the topology generation requires an enormous amount of time in order to complete. Consider the following case studies: In [7], for 11 components, the topology generation took between 45 to 218 seconds to complete, where the calculations were performed on a system using Intel Core i5-M540 @ 2.53 GHz and 8 GB RAM with Win7 and SWI-Prolog 7.4.2. (More on these results in section 2.2.3.) In [8], for the 16 components that they made use of, the topology generation was also completed somewhere under 300 seconds (i.e. under 5 min) with a system using 64-bit Intel(R) Core (TM) i7 @ 2.2 GHz and 8-GB RAM. The work done in [6] stands out from the previous two works by considering much more detailed components and requirements, and provides the best insight of all by showing how the computation time increases with the number of components. This is shown in Figure 2.3, where a single core of a system with Intel i7 7700HQ and 32GB of RAM were used. It can be seen how when selecting 11 components already 2.4 hours are needed to complete the topology generation. Selecting even 1 more component increases the waiting time to around two days. And from then on onwards the discussion quickly becomes about having to wait several years or even decades.

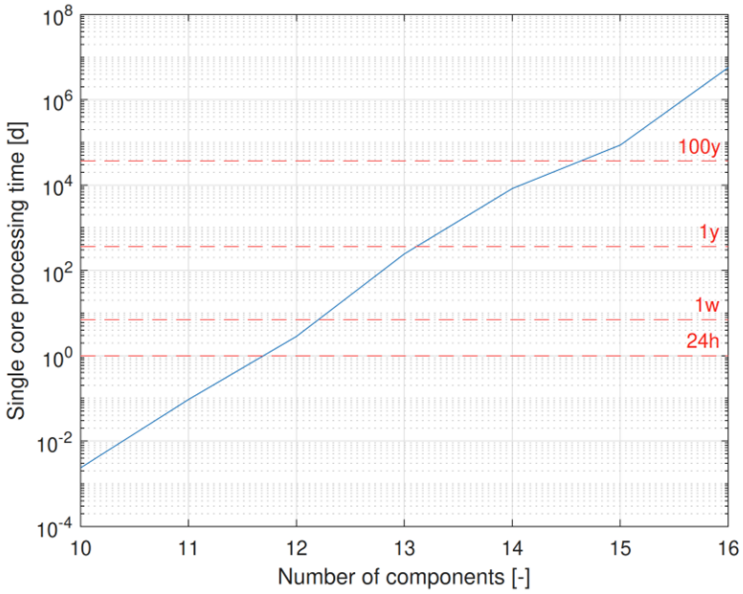


Figure 2.3: Figure from [6], showing how computation time on a log-10 scale increases with the number of components.

This is the main reason why for many studies within the field of Co-Design often only a small subset of only 4-10 TFs is selected [5]. An alternative is to join some of the components already together, e.g. joining a battery and an electric motor, in order to form one larger component. This is the approach used in [11], where a large set of components can be reduced to a smaller set of components before initiating the topology generation. In other papers such as [20, 21, 22] topology generation is omitted entirely, focusing instead on optimising a single topology. However, by disposing many of the TFs that are possible, all of these strategies sacrifice the most crucial advantage of Co-Design; namely the guarantee that at the end of the computations/design process, an optimal design is obtained.

As such research is needed that focuses on how to reduce the computational cost of topology generation. One starting point is to follow up on the suggestion made in [7], which was to look into using SAT solvers instead of the CSP solver that is used for the topology generation in [8, 7], as this might make more efficient computations. Whether this is true or what other alternatives exist to solve topology generation problems efficiently remains however unexplored.

2.2.3. PROBLEM 3: ISOMORPHIC TOPOLOGIES

In [7] a comparison is made between how long it takes to complete the topology generation when describing 11 components using a component-level-description (CLD) or a port-level-description (PLD).

In CLD each component is represented as a node that can be connected to other components using undirected edges. Meanwhile, in PLD, more detail is added to each component by describing the input and output ports it has. To connect these components, directed edges are used instead. In Figure 2.4 an example is shown of an electric motor and a battery described in CLD and in PLD, where in PLD it can be observed that the motor requires an input voltage and provides an output torque, whereas the battery only has one output that provides voltage.

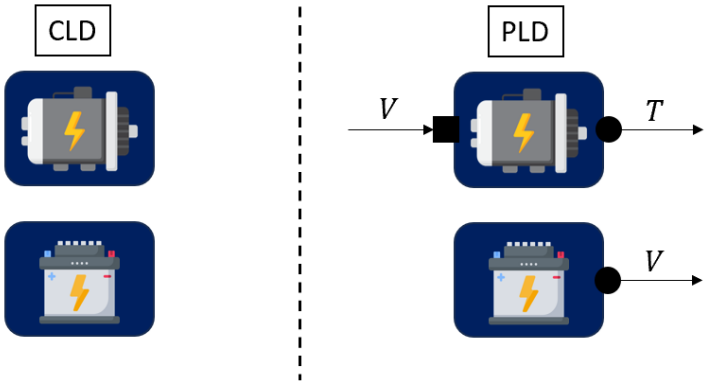


Figure 2.4: Example of a motor and battery described in a component-level-description (CLD), as is shown on the left, and a port-level-description (PLD), as shown on the right.

The reason these two description-approaches were developed in [7] was so that the topology generation performance could be compared. The hope was that with more detailed components, more constraints could be developed. This could then reduce the computation time to find all TFs, since fewer topologies remain to be explored. For example, using PLD to describe components allows for a port-to-port constrained to be expressed, in which any input port can only connect to one output port and vice versa. Such constraint does not exist when using CLD. Additionally, as also explained in [7], having specified for each component what their input and outputs are, the TFs that are generated from the topology generation directly also deliver input-output models that can be simulated. With CLD, the TFs still first need to be developed into models that only then can be simulated. It is therefore clear that there is a good reason to have components be described in more detail rather than less.

If one now refers to the results from [7], which are summarised in Table 2.2, then the expectations are only met partially. When using CLD to describe the 11 components, then 26 constraints could be developed, and using a system of Intel Core i5-M540 @ 2.53 GHz and 8 GB RAM with Win7 and SWI-Prolog 7.4.2, the topology generation could be completed in 45 seconds. On the other hand, when using PLD, indeed, as hoped, many more constraints could be developed. But quite contrary to what was expected, the topology generation was not completed any faster. Quite the opposite in fact; it took five times longer to complete the topology generation.

	CLD	PLD
Components	11	11
Constraints	26	168
Solving time [s]	45	218

Table 2.2: Summarised table of results from [7], showing how long the topology generation took to complete when using CLD or PLD.

Fortunately in [7] the reason for the significantly higher computation time could be identified. The issue lies with what is referred to in [7] as isomorphic topologies. These are topologies that albeit having different connections turn out to be identical. Figure 2.5 provides an example of two isomorphic TFs. Shown are two chains, where a battery feeds its voltage to an electric motor,

which in turn drives a generator, whose output voltage is used to charge a second battery. Crucially, the batteries B1 and B2 are actually two instances of the same component, i.e. B2 is a copy of B1. Thus whether the chain starts with B1 or B2 and ends with the other makes no difference. However, with the current methods that are employed, the CSP solver does consider these two TFs different.

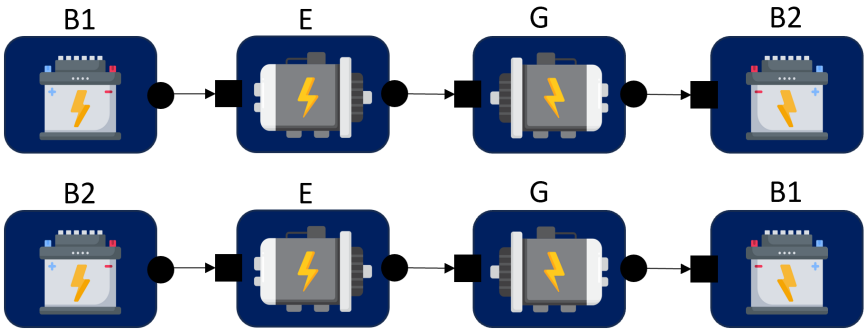


Figure 2.5: Example of two isomorphic TFs.

In [7] it was shown that when 11 components were used, in CLD a total of 21′536 isomorphic TFs were found. Using PLD instead, adding therefore more detail to the components, resulted in a total of 102′768 isomorphic TFs. Yet after post-processing these solutions to eliminate all isomorphic topologies and end up only with the set of TFs, in both cases almost the same number of TFs were found; 250 TFs for CLD and 262 TFs for PLD. This was used to indicate that increasing the amount of detail to describe the components results in an increased number of isomorphic topologies. Additionally, from the method that was applied to remove the isomorphic topologies, it was shown that isomorphic topologies also increase in number when the number of instances (i.e. copies) is increased.

However for several applications such as the HEVs considered in [5], it is needed that several instances of a component can be used. For example, one might be interested in exploring how HEVs with one single electric motor behave versus ones that have multiple ones. Additionally there often is a lot of data available about each component. Transitioning from a CLD description for the components to a PLD increases the use of the data that is available. And yet, with the current methods, having more instances available and having the components described in detail is being punished by significantly higher computation times for the topology generation process. No method seems to exist yet that can avoid producing these isomorphic topologies, which is why after the CSP solver has produced all TFs in [7], the results are post-processed in order to filter out all isomorphic TFs. (Adding therefore even more computational time to the topology generation process.) Research is therefore needed to see how one can avoid producing these isomorphic TFs in the first place, so that the CSP solver used for topology generation does not waste time searching for TFs it has already found. If such a method were developed, it should lead to a significant reduction in the required computational time to generate all TFs.

2.2.4. CONCLUSION ON LIMITATIONS

The problems that were described in sections 2.2.1 to 2.2.3 make it clear that the reason why Co-Design is limited to work on only 10-20 components is that with the current methods, topology generation demands an impossibly high computation cost once any more components are considered. However it is also clear that many open questions remain around the process of topology generation itself. Dedicating research into this process would lead to reducing the knowledge gap that exists on this topic. This knowledge may help to develop a topology generation method that is significantly faster and therefore requires a smaller computational cost.

2.3. RESEARCH QUESTION

From [5] it seems like most research has focused on topology optimisation, trying to circumvent the problems from topology generation in multiple ways such as described in section 2.2.2. In doing so though, much of the benefit of guaranteeing optimality that comes with Co-Design is sacrificed. Therefore instead research into topology generation is needed, as this would provide the understanding needed to possibly remove the 10-20 components limitation that originates from topology generation.

Focusing on topology generation, the most pressing problem seems to be the problem of isomorphic TFs described in section 2.2.3. Like visually summarised in Figure 2.6, instead of the topology generation process speeding up due to the increased number of constraints that can be placed thanks to describing the components in more detail (for example using PLD instead of CLD), the topology generation process actually slows down. This is because the added detail allows for an increased number of isomorphic topologies, whose negative impact is larger than the positive impact of an increased number of constraints.

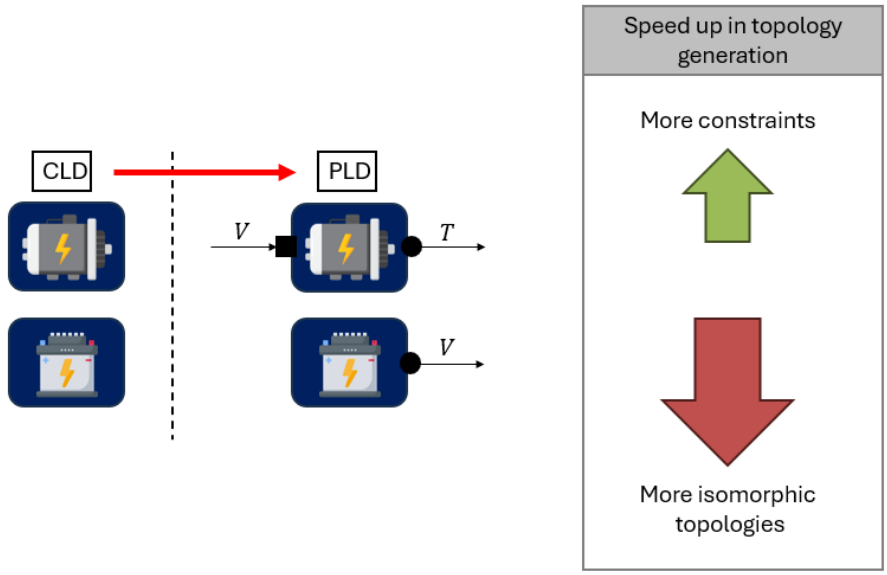


Figure 2.6: Summarising the problem with isomorphic topologies: When transitioning from a more basic description (such as CLD) to a more detailed one (PLD), more constraints can be placed, which should speed up the topology generation, but more isomorphic topologies are created, which slow the topology generation down [7].

As such it seemed most useful to dedicate research effort to the issue of the isomorphic topologies. The thought was that if a method could be put together that effectively avoids creating isomorphic topologies, the result of it being an additional constraint, then finally it may be possible to additionally benefit from the many more constraints that come with an increased level of component description. The result should be a significant speed up in the topology generation process, since the CSP solver would not need to waste computations on trying to find TFs it has already found, and has several more constraints due to the more detailed component description that allow it to find the TFs more quickly. These benefits will carry then over also to the topology optimisation process, since the optimisation space, the set of TFs, would be drastically smaller without any of the isomorphic topologies, and therefore would require less exploration to find the optimal TFs. Indeed, the entire Co-Design process would be influenced. Avoiding the isomorphic topologies may therefore be the key to removing the 10-20 components limitation

from Co-Design, and thus make it possible to apply Co-Design to develop the solutions needed to decarbonise the EU transportation sector.

Thus the following research question has become the focus of this thesis:

Main Research Question (MRQ)

In comparison to the current approach, how does avoiding isomorphic topologies, while using a detailed description of components, influence the time to complete generating all feasible topologies (TFs) when given a set of components and a set of requirements?

However answering such a research question requires a method that can compute TFs while avoiding isomorphic topologies. As explained in section 2.2.3, it seems such method does not yet exist. Therefore any candidate method that is put together and might be able to avoid producing isomorphic topologies will require at the very least to show that it is indeed able to do so in different use cases where such isomorphic topologies otherwise appear.

If this is done while using the same approach as is currently used for topology generation (see chapter 2), then for each use case all of the constraints required to find the TFs will have to be rewritten all over again, like explained in section 2.2.1.

What does stand out from the approach currently taken in topology generation is that even though nodes are used to represent the components and edges are used to represent the connection between the nodes/components [8, 5], no adjacency matrices are made use of, like one might expect when dealing with nodes and edges [23]. The work done in [6] is an exception, where indeed one large adjacency matrix was used to specify what component is connected to some component, i.e. the connection matrix. There already some constraints could be expressed in the form of what indices could not be used in the adjacency matrix.

It therefore seems useful to expand on the method that was used in [6] and see how this might help to reduce the number of constraints that need to be rewritten. Even if not all constraints can be rewritten into matrix form, with the ones where this is possible upper bounds can be expressed on the number of TFs that might exist with a set of components $\mathcal{S}_c$ and a set of requirements $\mathcal{S}_r$. Such information could provide the insight necessary to see how to speed up topology generation or how to avoid the isomorphic topologies. Therefore the following two supplementary research questions will be considered in this thesis as well.

Supplementary Research Question 1 (SRQ-1)

With respect to the current approach, how much does the use of adjacency matrices reduce the number of constraints that need to be rewritten?

Supplementary Research Question 2 (SRQ-2)

How much does each constraint used in topology generation restrict the number of feasible topologies (TFs) on a given set of components?

3

METHODOLOGY & RESULTS

In the previous chapter, chapter 2, it was explained how the current state-of-the-art approaches Co-Design and how its methods around topology generation are the reason it can only consider 10-20 components. After this review the research questions were presented.

In this chapter the methodology used to answer those research questions is described, while presenting the results that have been obtained along the way. The method that is put together here follows two key philosophies:

1. **Specify not *what* connections to place, but *how* to place connections.** In chapter 2 some examples of the constraints that are used by the current state of the art were presented. Of those one can distinguish between local and global constraints, i.e. constraints that apply only to some constraints versus constraints that apply to all components. In CLD many local constraints were placed that would specify to which other components a component could connect to, such as that an electric motor could not connect to a wheel directly but it could connect to a battery. In PLD however the global port-to-port constraint was placed, which specified that any port of any component could connect to only one port of any other component. It is the latter type of constraints that will be favoured, as this should lead to fewer but more effective constraints in comparison to multiple local ones. Additionally when the set of components changes, when using global constraints, these do not have to be rewritten.

2. **Maximise the use of the data that is available for each component.** As was explained in chapter 2, in CLD components are represented as nodes to which other nodes/components can connect to. However without a label under the node which specifies what component it is, there is no information that could help identify what the component is or what it does. This is also a reason why then local constraints are needed that specify which node can connect to which other. The lack of information about the components is compensated with the addition of local constraints. In PLD this is improved. Components can now be identified by their input and output ports, and whether the ports supply a voltage, torque or some other physical quantity. Even without a label one might already be able to guess what each component is. The goal will be to exploit as much of the data that is available for each component as possible, and thus the method builds on the PLD approach described in chapter 2.

The chapter starts off with section 3.1 by describing the basic setup that was used for which the rest of the work is built upon. Each section that follows builds on the work from the previous sections. The 3BT-3EM test, the first validation test described in section 3.1.3, is used as a case study to compare the results obtained in each section. In section 3.2 the set of potential topologies when no constraints are taken into account (TPs) is considered. In section 3.3 the port-to-port constraint from [7] is applied as a first constraint, to see how it reduces the set of TPs. This is carried on in section 3.4 with a second constraint, which specifies what ports are actually compatible with each other. Finally, in section 3.5, equipped with the knowledge and setup from the previous sections, the set of feasible topologies (TFs) is computed, where all constraints are taken into account. This also includes the proposed method that avoids the emergence of isomorphic topologies.

3.1. BASIC SETUP

In the following section the basic setup that was used is presented. First, in section 3.1.1 the modelling strategy is presented. This is followed with section 3.1.2, where an explanation of how the components are defined is provided, as well as how that data is set up and used. Since the goal is to develop a method that generates feasible topologies (TFs) without producing isomorphic topologies (see MRQ in section 2.3), in section 3.1.3 validation tests are described, which will be used to validate said method. Finally, in section 3.1.4, it is described how all of this has been implemented into a Python program with which the necessary computations are executed.

3.1.1. INTERCONNECTED SYSTEM MODEL FOR TOPOLOGY GENERATION

The Interconnected System Model (ISM) that is described in [24] is used as a basis for the method that is put together here. The model is visualised as a flowchart in Figure 3.1, where the subsystems $\Sigma_1,\dots,\Sigma_N$ collected in the top box provide the output vector $\boldsymbol{y} \coloneqq \left[\boldsymbol{y}_{\Sigma_1}..\boldsymbol{y}_{\Sigma_N}\right]^T \ : \ \boldsymbol{y} \in \mathbb{R}^q$. The output vector $\boldsymbol{y}$ is then fed into the gain $M \ : \ M \in \mathbb{R}^{p\times q}$, where the feedback matrix operation $\boldsymbol{u} = M\boldsymbol{y}$ is performed. The input vector $\boldsymbol{u} \coloneqq \left[\boldsymbol{u}_{\Sigma_1}..\boldsymbol{u}_{\Sigma_N}\right] \ : \ \boldsymbol{u} \in \mathbb{R}^p$ is then fed back to the subsystems $\Sigma_1,\dots,\Sigma_N$. The matrix M can therefore be used to define how the subsystems $\Sigma_1,\dots,\Sigma_N$ are coupled with each other.

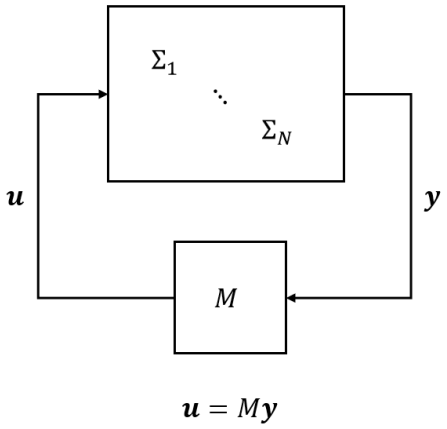


Figure 3.1: The Interconnected System Model (ISM), where the subsystems $\Sigma_1,\dots,\Sigma_N$ are fed the input $\boldsymbol{u}$: $\boldsymbol{u} = M\boldsymbol{y}$, and provide the outputs $\boldsymbol{y}$.

Define each component in $\mathcal{S}_c$ as a subsystem in the ISM. Additionally define the matrix M to be boolean, i.e. $M \in \mathbb{B}^{p\times q} \ : \ \mathbb{B} \coloneqq \{0,1\}$. Then for each $m_{i,j}$ in M, it specifies whether there exists a connection between the output y_j and the input u_i. In other words, the matrix M can be used to specify the connections that exist between components, i.e. the topology.

Because M is a boolean matrix, there exists a finite set of potential topologies $\mathcal{T}_p \coloneqq \{M_1..M_{pq}\}$, where with $\boldsymbol{y} \in \mathbb{R}^q$ and $\boldsymbol{u} \in \mathbb{R}^p$, the number of potential topologies (TPs) is $|\mathcal{T}_p| = 2^{pq}$. However, as is shown in section 3.3 to section 3.5, many connections that exist in M can actually not be placed. This reduces the number of TPs that exist. Additionally, the goal is to use the set of components $\mathcal{S}_c$ in order to meet a set of requirements $\mathcal{S}_r$. Topologies that achieve this are referred to as feasible topologies (TFs), such that $\mathcal{T}_f \subseteq \mathcal{T}_p$.

In [8, 7] additional boolean variables exist that specify whether a component/subsystem is selected or not. Here however it will be assumed that a component is selected if a connection has

been established between it and some other component, just like it was done in [6]. The reasoning is that it makes little sense to establish a connection between two components if any of those two components are not present. The booleans $m_{i,j}$ contained in M are therefore all the booleans that will be considered.

It is worth mentioning that the ISM resembles a lot to the typical feedback loop used in control engineering. The main difference is that while usually the feedback gain M is such that $M \in \mathbb{R}^{p\times q}$, here M is a boolean matrix such that $M \in \mathbb{B}^{p\times q}$. Additional constraints that specify which connections can be placed restrict M even further. As such, when trying to obtain the ideal topology from a set of components $\mathcal{S}_c$ to meet a set of requirements $\mathcal{S}_r$, this is like designing the ideal feedback gain M, but with the challenge that M is extremely restricted in the values it can take.

The ISM also helps to differentiate between different requirements that can be placed. Three different requirements have been defined here:

- **Output requirements:** From the set of components $\mathcal{S}_c$ a certain output $y \in \boldsymbol{y}$ may be required. If a single output is required, then the TFs require a component that can deliver this output.
- **External input requirements:** Besides the inputs components may receive from other components through the boolean matrix M, some external inputs may need to be taken into account as well. For a single external input, the TFs require then a component to which this input can be fed to.
- **Global constraints requirements:** Unlike the previous two requirements, for which some component needs to be selected to deliver the correct output or to which some external input can be fed to, this requirement acts on all components of any TF. An example is a constraint on the total mass of the components in a TF.

More requirement types may exist, but this is beyond the scope of this thesis. In fact, in this work, only a single output requirement is going to be considered. This is sufficient to answer the main research question considered in this thesis, as is presented in section 2.3.

3.1.2. COMPONENT DESCRIPTION AND COMPOSITION

In the following section the description used for the components is defined. This description builds upon the PLD description used in [7] (see section 2.2.3) following the recommendations also mentioned in [7]. Inspiration has also been taken from the physical network modelling approach that is used in Simscape from Mathworks [25]. (Refer to my literature review, section 4.2, on physical network modelling, for more information on this.)

The intention was very much to set up a description that goes beyond the PLD used in [7], and could be expanded in the future to maximise the use of the data that is available about each component. The reason to directly start off with a detailed description rather than a lesser one is so that the emergence of isomorphic topologies cannot be ignored. (As was described in section 2.2.3, using a more detailed component description leads to more isomorphic topologies.)

The component description used here, which shall be referred to as the property-port level description (PPLD), starts off by defining properties. These are analogous to the input and output variables that are typically used in state-space or transfer function models. In Figure 3.2 an example is provided of the PPLD applied to a set of electric motors. The properties are shown as the circles with V, m, T or ω in them. It is also assumed that the components are modelled as fully observable systems. That means that also the states of the components are available as outputs.

3.1.2.1. NODE, UNIT AND DOMAIN TYPES

The properties are distinguished by the set each property belongs to for each type that exists. To start off, there are three types:

- Node type, which specifies whether the property is an input or an output. In Figure 3.2 the V property is an input property, while the m, T and ω properties are all outputs.
- Unit type, which specifies what variable the property is. The V property in Figure 3.2 is a voltage, m is a mass, T is a torque, and ω is a rotational speed.
- Domain type, which specifies to what physical domain the property belongs to. The V property in Figure 3.2 belongs to the electrical domain, while the rest of the properties belong to the mechanical one.

To define the types with a bit more detail, consider as an example the node type. All properties will belong to either a set of input properties $\mathcal{U}$ or to a set of output properties $\mathcal{Y}$. Therefore the input and output set are disjoint, i.e. $\mathcal{U} \wedge \mathcal{Y} = \emptyset$. The two disjoint sets represent the (discrete) node type, which is denoted using angle brackets, i.e. $\langle \mathrm{N} \rangle := \{\mathcal{U}, \mathcal{Y}\}$.

While the node type is defined by at most two different sets, the other types may have many more sets. This depends on the set of components that is being used. For example in Figure 3.2, if no further components are considered, then the unit type is defined by the four variables V, m, T and ω. As such, if the set of components changes, then so do the types.

3.1.2.2. USE IDS TO RECORD PROPERTIES AND TYPES

To keep track of all of these elements, every property and set (for the types) is given a unique ID and saved into a list. The lists are denoted here using square brackets. For example, the list of property IDs is written as [P-ID]. Similarly three other lists can be written for the node, unit and domain type, i.e. [N-ID],[V-ID],[D-ID]. For example the list for the node type [N-ID] is of length 2, containing one ID for the input set $\mathcal{U}$ and another ID for the output set $\mathcal{Y}$.

The IDs that are used are universally unique lexicographically sortable identifiers, otherwise known as ULIDs. As described in [26], these are similar to the more commonly used universal/global unique identifiers (UUID/GUID), which use the current time stamp and random generators to create unique IDs. ULIDs however have the extra feature that they can be sorted by the order in which they have been created. The ability to order components is an essential ingredient for the method that is used to avoid isomorphic topologies, as will be explained in section 3.5.

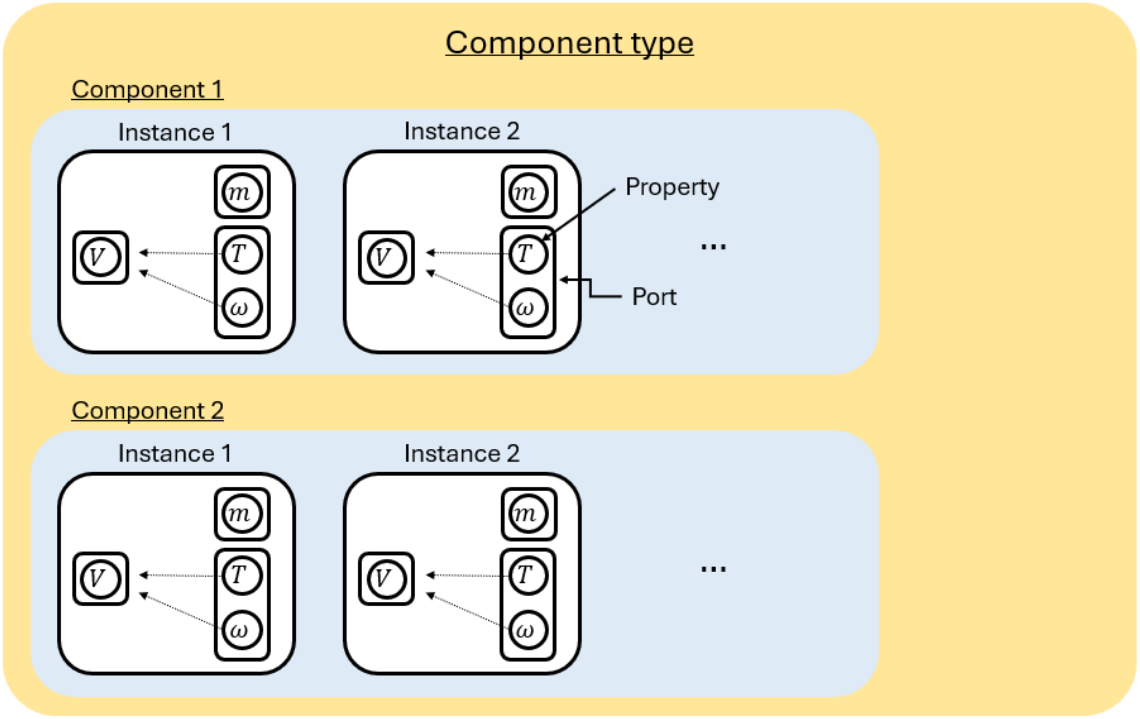


Figure 3.2: The component description used in this thesis, referred to as the Property-Port Level Description (PPLD), here showing an example of the description for a set of electric motors.

3.1.2.3. PORTS, INSTANCES, COMPONENTS AND COMPONENT TYPES

Once the properties have been defined by these three basic types, they are assembled by further types to define ports, instances, components and component types.

Ports will contain at least one property. These are shown in Figure 3.2 as the rounded rectangles that contain the circles V, m, T and ω. The idea behind ports is that in some cases, if a certain output variable is requested, it cannot be requested without also accepting another output variable. For example with the electric motors in Figure 3.2, if one chooses to use the torque output that is provided by the electric motor, then this will be provided with the output ω as well.

Additionally, it is assumed that ports can only have either input properties or output properties, but not properties with mixed node types. That implies that ports can be distinguished as well between input ports and output ports.

With the ports defined, instances of components can be formed. The idea behind instances is to allow to make copies of a component. In Figure 3.2 two components are shown. Each component represents a different electric motor. However when considering only one component, all instances represent the same electric motor. As such, each component has at least one instance, and each instance has at least one port.

Here the assumption is used that the output ports and input ports for each instance need to be distinguishable and therefore have unique types for their properties. As such instances/components that would have multiple ports that all provide the same node, unit, and domain type are ignored. (For example multiple voltage outputs.)

Finally, components can be grouped into component types if they are similar enough. This is a strategy that was employed in [27, 20, 22], where multiple components, for example electric motors, are grouped into one type. However in [27, 20, 22] it is not defined when two different components are similar enough to be put into the same component type. Instead, this is defined apriori. Such approach is avoided here, as this would equate to defining local constraints and would therefore require to be rewritten for each new set of components one might consider. Instead here components are grouped together into component types when the components turn out to have the same unit types.

The PPLD assembled here is presented as a UML class diagram in Figure 3.3. The diamond arrows represent composition relationships. Here they indicate for example that at least one property needs to exist before 1 port can be created. (And no port will exist with no properties present.)

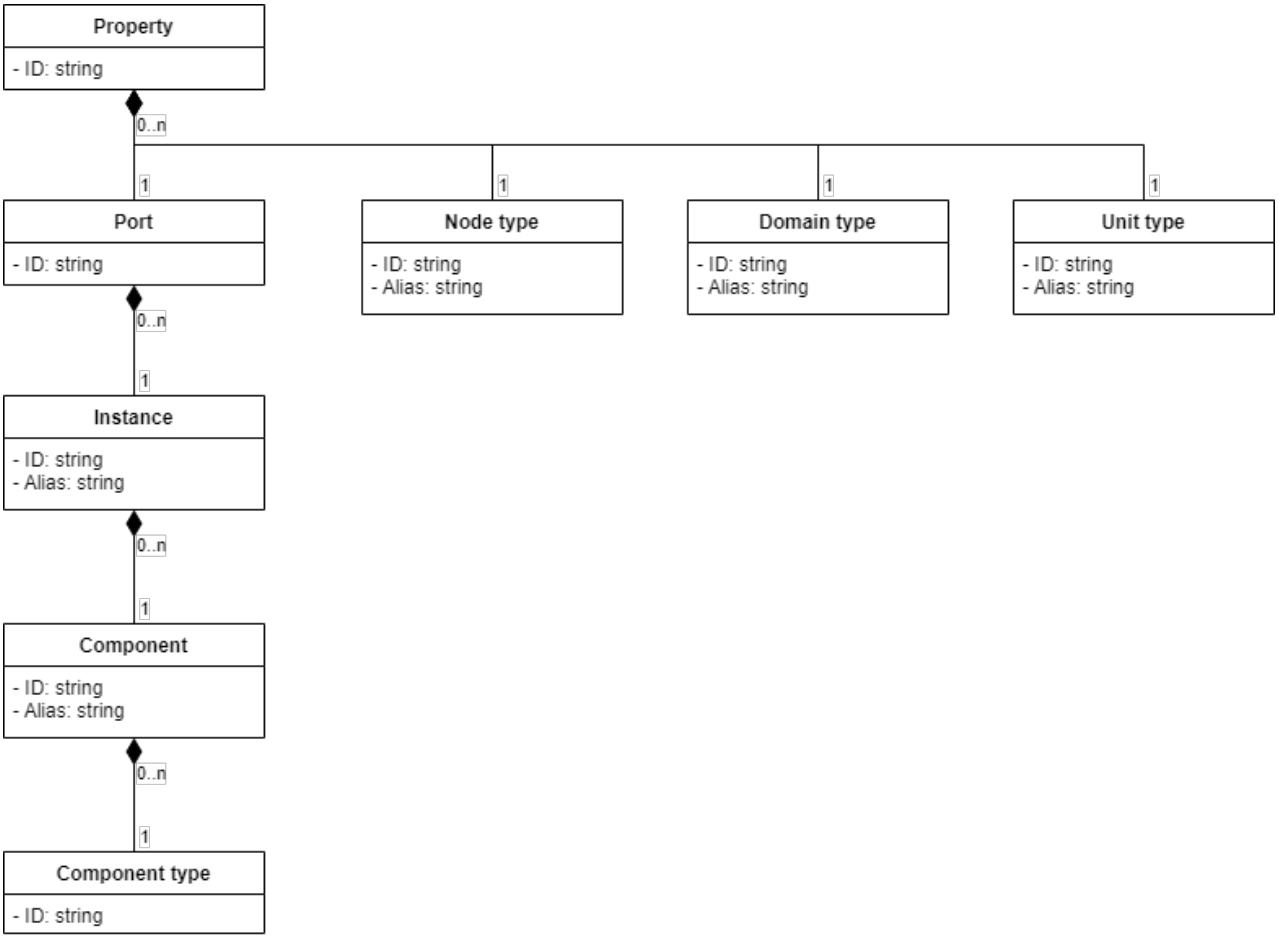


Figure 3.3: The UML class diagram for the Property-Port Level Description (PPLD).

Just like with the properties and the node, unit and domain types, also for each port, instance, component and component type a unique ULID is given. This is shown in the attributes of the UML class diagram in Figure 3.3. In total therefore eight different ID lists are made use of:

- [P-ID] : Ordered list/vector of property IDs.
- [T-ID] : Ordered list/vector of component type IDs.
- [C-ID] : Ordered list/vector of component IDs.
- [H-ID] : Ordered list/vector of instances.
- [R-ID] : Ordered list/vector of port IDs.
- [N-ID] : Ordered list/vector of node types.
- [V-ID] : Ordered list/vector of unit types.
- [D-ID] : Ordered list/vector of domain types.

For some of these lists a list aliases is also provided. These also appear as attributes in the UML class diagram in Figure 3.3. For example components are supplied an alias next to their ULID. This could be for instance an alias to specify that a component is a lithium battery. These aliases are merely there to make it easier to review results and debug problems in the code. (More on this in section 3.1.4.) It is always the ULIDs that are used as reference.

3.1.2.4. DEFINING MEMBERSHIPS AND COMPOSITIONS

Besides defining IDs for every element in the PPLD, it must also be specified where each element belongs. (For example which properties belong to some specific port.) To achieve this, unweighted adjacency matrices (UAMs) are used to describe the membership between two elements. This approach is taken directly from the methods used in fuzzy logic, where the membership UAMs are also referred to as relations [28]. The following membership UAMs are defined:

- μ([P-ID] × [T-ID]) : Membership of each Property ID with each component type.
- μ([P-ID] × [C-ID]) : Membership of each Property ID with each component ID.
- μ([P-ID] × [H-ID]) : Membership of each Property ID with each instance.
- μ([P-ID] × [R-ID]) : Membership of each Property ID with each port ID.
- μ([P-ID] × [N-ID]) : Membership of each Property ID with each input/output type.
- μ([P-ID] × [D-ID]) : Membership of each Property ID with each domain type.
- μ([P-ID] × [V-ID]) : Membership of each Property ID with each unit.

Here the cross product between one list and another indicates that a UAM is formed. The symbol in front of the cross product indicates what the relation between the two lists is. In this case, with the symbol μ, it is indicated that the relation refers to a membership. For example, for μ([P-ID] × [T-ID]) it is specified to which component type in [T-ID] each property in [P-ID] belongs to.

Defining membership with this strategy allows that (fuzzy) composition like described in [28] can be used. (Refer to appendix A.1 to review the definition of a fuzzy composition.) Consider the case where the relation μ([T-ID] × [C-ID]) needs to be used. In (3.1) it is then shown how this

can be obtained from the membership UAMs that were defined earlier, where the symbol '$\circ$' is used for the fuzzy composition operator.

$$\mu([\text{P-ID}] \times [\text{T-ID}])^T \circ \mu([\text{P-ID}] \times [\text{C-ID}]) = \mu([\text{T-ID}] \times [\text{C-ID}]) \tag{3.1}$$

Composition can also be used to extract elements that meet certain conditions. Consider as an example that we would like to find all properties whose node type is an input, and that belong to some instance n. The following steps are taken:

1. Select properties whose node type is an input:

   $$\mu([\text{P-ID}] \times [\text{N-ID}]) \circ \mu([\text{N-ID}(u)]) = \mu([\text{P-ID}|\text{N-ID}(u)])$$

   Here $\mu([\text{N-ID}(u)])$ provides a boolean vector of length $|[\text{N-ID}]|$, with a 1 for the selected set $\mathcal{U}$, here specified by the bracket expression (u).

2. Select properties that belong to some instance n:

   $$\mu([\text{P-ID}] \times [\text{H-ID}]) \circ \mu([\text{H-ID}(n)]) = \mu([\text{P-ID}|\text{H-ID}(n)])$$

3. Use AND operation to find all properties whose node type is an input and that belong to the instance n:

   $$\mu([\text{P-ID}|\text{H-ID}(n)]) \wedge \mu([\text{P-ID}|\text{N-ID}(u)]) = \mu([\text{P-ID}|\text{H-ID}(n).\text{N-ID}(u)])$$

   The period symbol '.' is used as a shorthand here for the AND operation usually expressed by the wedge symbol $\wedge$, in order to make sum-of-product expressions more readable. (See section 2.6.1 on sum-of-products and product-of-sums in [29] for the definition of these boolean expressions.)

Composition is therefore an essential operation in order to manipulate and retrieve the data that is available for any set of components $\mathcal{S}_c$.

Luckily, it turns out that with the use of UAMs, which are boolean matrices that can only take as values either True or False (1 or 0), the fuzzy composition can be simplified to a regular matrix multiplication.

Proof. Let A and B be two boolean matrices such that $A \in \mathbb{B}^{p\times q}$ and $B \in \mathbb{B}^{q\times r}$, with $\mathbb{B} := \{0,1\}$. The matrix multiplication of A and B would result in another matrix C such that:

$$c_{i,j} = \sum_{k}^{q} a_{i,k} b_{k,j} \tag{3.2}$$

Only three types of $a_{i,k}b_{k,j}$ products exist:

- $0 \cdot 0 = 0 = \min(0,0)$
- $1 \cdot 0 = 0 \cdot 1 = 0 = \min(0,1)$
- $1 \cdot 1 = 1 = \min(1,1)$

When taking the sum of these products, for the first two cases the sum will always lead to zero. For the third case, if such a product would appear more often when multiplying A and B, then a regular sum would grow to some value ≥ 1. However with boolean variables, $1+1+\cdots=1=\max(1,\ldots,1)$.

Therefore, the matrix multiplication of A and B leads to another boolean matrix C, where the matrix multiplication in (3.2) can be expressed using max and min terms instead, like shown in (3.3):

$$c_{i,j} = \max_{k \in K} \min(p_{i,k}, q_{k,j}) \tag{3.3}$$

This expression is identical to the max-min composition from described in [28]. (See appendix A.1). ■

This is helpful, since for programming packages/libraries that offer functions to make matrix computations are widely available. For example in Python, packages like Numpy, Scipy or Pandas already provide functions to multiply two matrices together, and where one can specify that these matrices are boolean, therefore making composition possible.

Still, to distinguish both from a more general fuzzy composition operation and otherwise from a regular matrix multiplication operation, the matrix multiplication of two boolean matrices will be referred here as a boolean composition.

3.1.2.5. LOCAL MAPS AND CONNECTION MATRIX

Two last elements remain to complete the PPLD: Defining which input properties each output property depends on, referred here as the local map, and defining all potential connections that can be made between components, ignoring all constraints. The local map can also be understood as the internal connections of components that are already set in place, while the potential connections refer to the external connections between components that still need to be set in place.

To make these definitions, it is helpful to take a look at the potential connections UAM $\nu([\text{R-ID}]_{[\text{H-ID,N-ID}]} \times [\text{R-ID}]_{[\text{H-ID,N-ID}]})$, which shows the potential connections that could be made from port to port, only taking into account the PPLD definitions that have been made so far. Here the symbol ν is used instead of the symbol μ in order to indicate that the UAM here represents all the potential connections (instead of the memberships). Additionally, the expression $[\text{R-ID}]_{[\text{H-ID,N-ID}]}$ indicates that the list of port IDs [R-ID] is not sorted by itself, but sorted first by the order of [N-ID], and then by the order of [H-ID]. This is referred here as a multi-order, and in this case uses the membership UAMs $\mu([\text{R-ID}] \times [\text{N-ID}])$ and $\mu([\text{R-ID}] \times [\text{H-ID}])$ as reference for the order. The potential connections UAM $\nu([\text{R-ID}]_{[\text{H-ID,N-ID}]} \times [\text{R-ID}]_{[\text{H-ID,N-ID}]})$ is visualised in Figure 3.4.

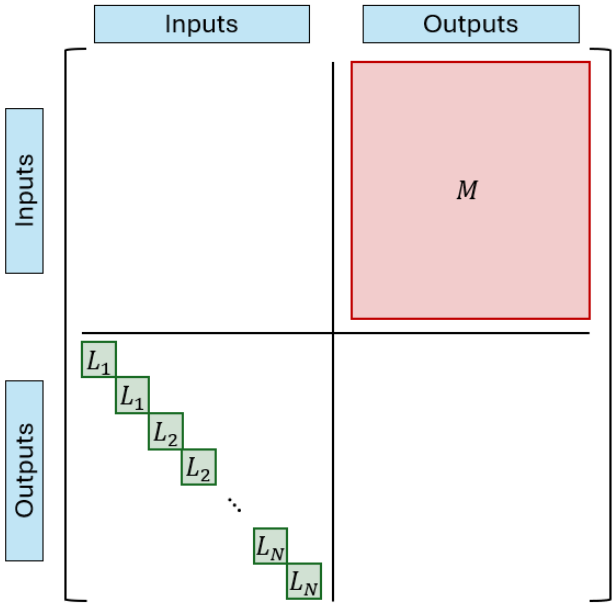


Figure 3.4: Visualisation of the potential connections between ports as expressed by $\nu([\text{R-ID}]_{[\text{H-ID,N-ID}]} \times [\text{R-ID}]_{[\text{H-ID,N-ID}]})$.

As can be seen from Figure 3.4, the block matrices [Input] × [Input] and [Output] × [Output], referring to $\nu([\text{R-ID}|\text{N-ID}(u)] \times [\text{R-ID}|\text{N-ID}(u)])$ and $\nu([\text{R-ID}|\text{N-ID}(y)] \times [\text{R-ID}|\text{N-ID}(y)])$ respectively, are empty. This is because it is insisted upon that inputs connect to outputs and vice versa, i.e. no input port can directly connect to another input port, same for output ports.

Also the block matrix [Output] × [Input] is mostly empty, where the block matrix refers to $\nu([\text{R-ID}|\text{N-ID}(y)]_{[\text{H-ID,N-ID}]} \times [\text{R-ID}|\text{N-ID}(u)]_{[\text{H-ID,N-ID}]})$. Only the diagonal is filled with the local maps for each instance considered in the set of components $\mathcal{S}_c$. Here the case is considered in which there are two instances per component, thus showing two identical local maps for each

component, before moving to the next component/local maps. Each local map defines a membership UAM that specifies on which input properties the output properties depend. In other words, each local map L_n is defined as shown in (3.4).

$$L_n := \mu([\text{P-ID}|\text{C-ID}(n).\text{N-ID}(u)] \times [\text{P-ID}|\text{C-ID}(n).\text{N-ID}(y)]) \tag{3.4}$$

Given that the block matrix [Output] × [Input] is so sparse, it makes more sense to only save the individual local maps $[L_1..L_N] : N = |[\text{C-ID}]|$. To be able to retrieve the local maps, ULIDs are assigned to each of them, and the membership UAM $\mu([\text{P-ID}] \times [\text{L-ID}])$ is assembled. Thus properties of a component can be used in order to retrieve the local map for said component.

The final block matrix, [Input] × [Output], labelled here as the M matrix, is indeed the same matrix as in the ISM. (See Figure 3.1.) The block matrix refers to $\nu([\text{R-ID}|\text{N-ID}(u)]_{[\text{H-ID,N-ID}]} \times [\text{R-ID}|\text{N-ID}(y)]_{[\text{H-ID,N-ID}]})$. Because it is a full block matrix, how the columns or rows are ordered does not matter, and thus the block matrix can be expressed as shown in (3.5). It represents all potential connections that can be placed, where the connections are placed between the ports that have been defined in the PPLD.

$$M := \nu([\text{R-ID}|\text{N-ID}(u)] \times [\text{R-ID}|\text{N-ID}(y)]) \tag{3.5}$$

As such, to study topology generation, only the matrix M needs to be considered. Constraints will be placed that restrict what forms the matrix M can take.

Define additionally a reduction operation on UAMs, which gets rid of all rows and columns that only contain zeroes. This will be denoted as a small subscript by the letter 'r'. Then the number of input ports R_u and output ports R_y can be defined as shown in (3.6).

$$\begin{aligned} R_u &:= \mu_r([\text{R-ID}|\text{N-ID}(u)]) \\ R_y &:= \mu_r\left([\text{R-ID}|\text{N-ID}(y)]\right) \end{aligned} \tag{3.6}$$

Then M can also be expressed as a boolean matrix like shown in (3.7):

$$M : \ M \in \mathbb{B}^{R_u \times R_y} \tag{3.7}$$

3.1.3. VALIDATION TESTS

The goal of the method that is put together here, as described in section 2.3, is to retrieve the set of TFs $\mathcal{T}_f$ from a set of components $\mathcal{S}_c$ and a set of requirements $\mathcal{S}_r$. This was meant to be done while avoiding the emergence of isomorphic topologies. To validate the method, three tests have been put together, which are introduced in this section.

3.1.3.1. THE 3BT-3EM TEST

The 3BT-3EM test refers to a set of components that makes use of three batteries as components, and three electric motors. See Figure 3.5 as an example. There are two instances for every component that is used, totalling to a sum of six instances. The 3BT-3EM test however allows also more instances to be specified per component type. As such, the set of components shown in Figure 3.5 is more specifically referred to as the 3.2BT-3.2EM test.

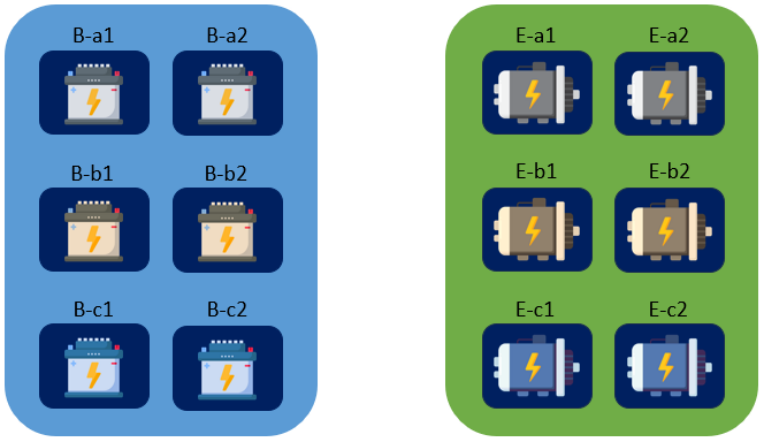


Figure 3.5: The reference set of components, which uses 3 components with 2 instances each for two component types, batteries and electric motors.

A different way of representing the set of components is by specifying instead the component types, and what the functionalities and requirements are for each. This representation is used in [20, 27, 21, 22], and is referred to here as a Functionalities-Requirements-Diagram (FRD). Figure 3.6 shows the FRD for the 3BT3EM test. It specifies two component types, batteries and electric motors, but does not specify the number of components or instances that exist in each type. Each component type shows what it requires on the left-hand side, in the red box, and what it supplies on the right-hand side, in the green box. In each box the ports are shown, where each square-bracket expression is a single port. As can be seen, both the batteries and electric motors have four output ports and one input port. The electric motors also have ports with two properties in them, like for example the output port '[Torque,Omega]'. Here the properties are specified by their unit type, to make the FRD more readable, even though properties are distinguished by more types and unique ULIDs. The arrow between the batteries and the electric motors specifies that there is a compatible port between the two component types, namely '[Voltage]', meaning in this case that the batteries can supply the voltage requirement of the electric motors.

The difference between Figure 3.5 and Figure 3.6 is that in the latter the number of components and instances is not specified apriori. Instead different numbers of components and instances can be specified on the go, creating different sets of components, and therefore also different tests that still have the same FRD. That way the method can be tested in multiple ways by only

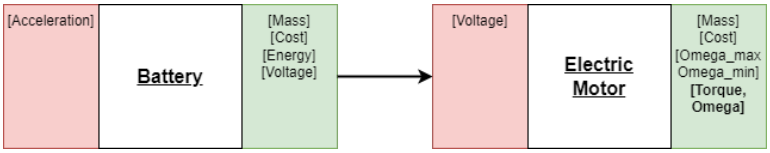


Figure 3.6: FRD of two component types, batteries and electric motors.

changing the number of components and instances, and not requiring new component types to be included. This makes organising the tests easier, and helps in predicting how many TFs one should expect for each test.

The final element in the FRDs is that not only are the component types for the set of components $\mathcal{S}_c$ specified, but also the set of requirements $\mathcal{S}_r$. Specifically, in the work done here, as a first step for the set of requirements $\mathcal{S}_r$ only a single output requirement is specified, as this is sufficient to answer the main research question. (See section 3.1.1 for the different types of requirements that have been laid out. The main research question is presented in section 2.3.) In Figure 3.6 the port '[Torque, Omega]' is shown in bolt, indicating that this is the output requirement that is being specified. Therefore one can see how requirement chains can be formed, where when specifying that '[Torque, Omega]' is needed as an output, the electric motors are chosen, but these require a voltage input, which the battery can provide.

While different tests can be assembled with the same FRD, it is important to state that the method will not have the component types defined apriori, nor will the arrows be specified. It is still required that the method finds out which components are of the same type, and to find what ports are compatible as indicated by the arrows.

The 3BT-3EM test consists of keeping the number of components fixed, while the number of instances is increased. Passing the 3BT-3EM test means that:

- The number of TFs remains constant at nine different topologies, i.e. no isomorphic topologies are produced.
- If no isomorphic topologies are produced, then increasing the number of instances should have a minor impact on the computation time compared to the computation time when isomorphic topologies are not rejected, as it is the case with the current methods. As was explained in section 2.2.3, rejecting isomorphic topologies would mean that no computation time needs to be wasted on trying to find TFs that have already been found.

3.1.3.2. THE ISO-CAR TEST

To truly test the method on whether it can avoid isomorphic topologies, a second test is needed. This is because in the first test, the 3BT-3EM test, one could avoid the isomorphic topologies by only considering the first instance of each component, ignoring all other instances.

However, as was discussed in section 2.2.3, there are plenty of cases where multiple instances need to be considered. The ISO-car test is such a case.

Figure 3.7 shows the FRD of the ISO-car test, showing that three component types are considered for the set of components, and that '[Thrust]' is specified as an output requirement. The unique feature of the CAR component type is that it requires two batteries: One that is directly connected to the car, and a second battery to power the electric motor that drives the car.

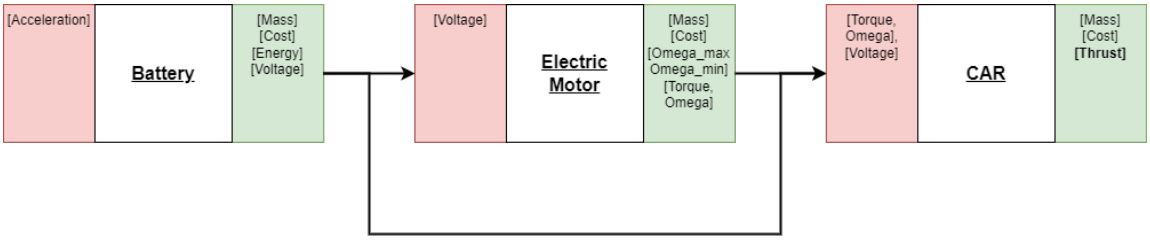


Figure 3.7: FRD for the ISO-car test, which consists of a car that requires two batteries: One directly connected to the car, and another driving the electric motor of the car.

In the simplest case, a single battery with two instances is used, while only one electric motor and one car is used. The expectation for the TFs is that:

- Only one TF is produced.
- The TF uses a first instance for the battery that is directly connected to the car, and a second instance for the battery that drives the electric motor.

As was explained in section 3.1.3.1, more components/instances can be used to modify the ISO-car test and test the method also for those cases.

3.1.3.3. THE LOOPS & BRANCHES TEST

In the final test, the method is tested for an application. Given that as a first challenge the truck industry can be considered, the small Goupil G4 truck was used as a first case study. The Goupil G4 truck is used among other by the groceries delivery company PicNic, based in Netherlands, where TNO already is assisting them with their research.

In Figure 3.8 the FRD for this test is presented. The component types chosen for this test were largely inspired by the main components outlined in the Goupil-G4 brochure [30].

Two crucial features exist in the FRD for this test that did not exist in the previous two tests. The first one is that in this test, the driver is considered a component as well. It acts however as a controller. This leads to feedback loops. The second feature is that the tractor[1] can either be powered directly by an electric motor, or will first require a gearbox before connecting to the electric motor. This is referred to as a branching. This is why this test is referred to as the Loops & Branches test.

In the simplest case, only one component and instance is available for each component type. The expectation is that:

- There are only two TFs, namely a tractor that does use the gearbox, and another that does not.
- The driver/controller is connected correctly. Among other, it should feed back to the tractor.

As was explained in section 3.1.3.1, more components/instances can be used to modify the ISO-car test and test the method also for those cases.

[1] Trucks are usually composed of two units: The tractor, and the trailer that gets pulled by the tractor. The tractor is therefore the driving unit.

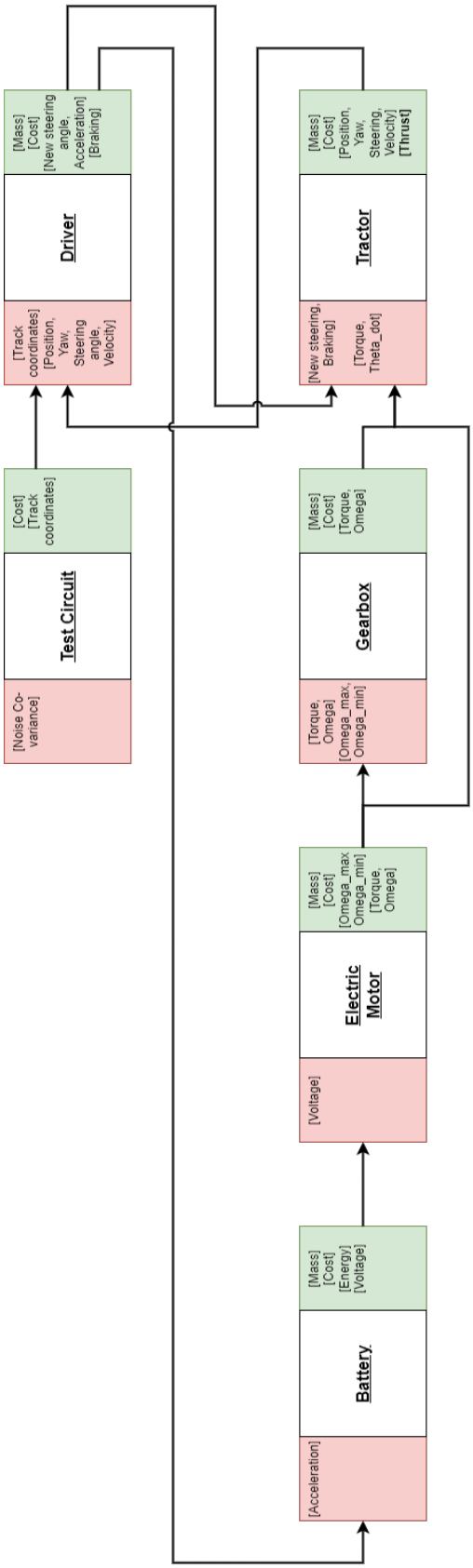


Figure 3.8: FRD for the Loops & Branches test, where components types for a PicNic truck are considered. This includes a driver as a component type in order to include a controller in the set of components.

3.1.4. TOPOGEN PYTHON PROGRAM

The method that is put together here is implemented into a program. The program was given the name of 'TopoGen', as an abbreviation for topology generation. The program was developed in the Spyder Integrated Development Environment (Spyder IDE) using Python 3.12, and was run on a Windows 11 machine using a 11th Gen Intel(R) Core(TM) i7-1185G7 @3.00GHz (4 Cores 8 Logical Processors) processor and 16 GB of RAM. Additionally the AI Perplexity (perplexity.ai) was used to help develop the code[2].

TopoGen is made available under the private GitHub repository `mdelgadoschwartz/TopoGen`. At the moment no license exists for this project. By default, this means that under copyright law all rights are reserved to me, Marco Delgado (Gosálvez) Schwartz, the creator. For potential research contributions or collaborations please contact me under the email marco@delgado-schwartz.com.

Figure 3.9 shows the flowchart of the main steps that are taken by in TopoGen. Two main processes are carried out: First the option is given to either select an existing set of components or create a new one. A library of components is available, with which new sets of components can be created. Once a set of components has been selected, the next process is carried out. In this step, different topology generation calculations can be executed. These are:

- Calculate the number of potential topologies with no constraints (TPs).
- Calculate the number of potential topologies with bijective connections (TPBs).
- Calculate the number of potential topologies with bijective compatible connections (TPBCs).
- Generate feasible topologies (TFs) based on requirements.

What happens in each step is explained in more detail in the following chapter, chapter 3. TopoGen allows the user to repeat any of the processes as many times as needed before exiting the program.

[2]Perplexity works similar to Chat-GPT, but with an emphasis on accuracy. Among other this is achieved by providing the sources that were used to produce the answers. See more at `https://www.perplexity.ai/hub/faq/what-is-perplexity`.

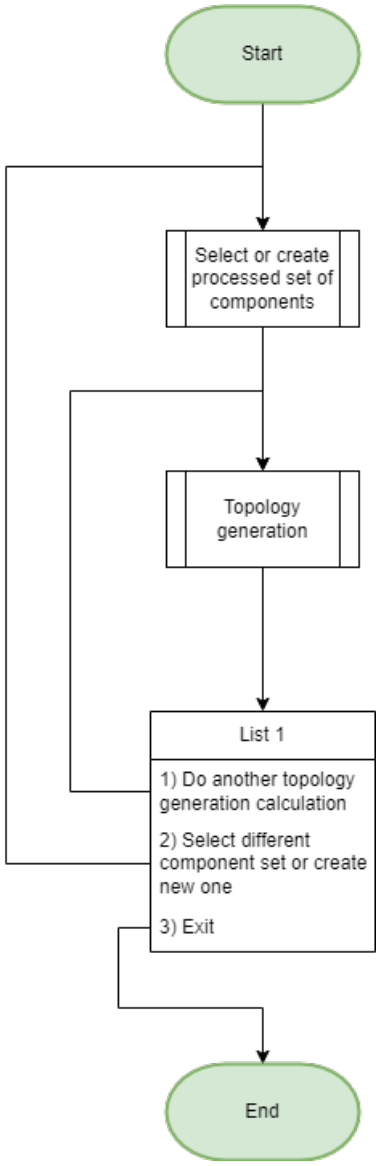


Figure 3.9: Flowchart that describes the main steps the TopoGen program takes from start to finish.

Figure 3.10 shows an overview of the folders and files of the GitHub repository. As is standard practice, a `README.md` file is provided in the main folder, which introduces any visitor to the project and program structure. Additionally a flowchart can be found in `Flowchart_Program.drawio`, which describes the key steps taken by the TopoGen program. The `main.py` file runs the entire program. (To view the variables that are created in TopoGen, it is recommended to use an IDE like the Spyder IDE.)

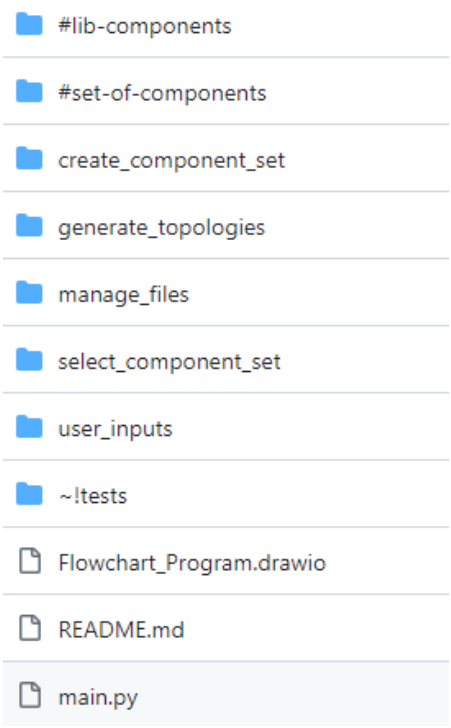


Figure 3.10: Overview of folders and files for the TopoGen program.

The `#lib-components` folder is the folder in which the description for components is available. For each component an Excel file exists, which describes the properties of the component using the PPLD setup that was described in section 3.1.2. If the user chooses to create a new set of components, they can specify which components they would like to use from the components available in `#lib-components`, and then specify how many instances they would like to have for each selected component. TopoGen then processes that selection to form a set of components, which involves creating the ID lists and membership UAMs as was described in section 3.1.2. Any set of components that is created is saved in `#set-of-components` as pickle file, which allows the user to reuse some set of components again in the future.

TopoGen is run by user inputs. In the folder `user_inputs` the functions used to create different user inputs are available. Whenever a request is made to the user, TopoGen will process the input to see if it is compatible with the request. If a mistake has been made, it will provide a help message. The help message can also be accessed directly by typing 'help' in the command line. Once Enter is pressed, the request is made again, allowing the user to adjust their response. Additionally, should it be necessary, the user can type 'exit' to stop the program.

The folder `~!tests` contains the tests that were executed to produce the results that are shown in chapter 3. The rest of the folders contain the functions that complete the rest of the TopoGen program.

TopoGen has been put together in such a way that every function is available as its own file. In each file, besides defining the function, also a test is provided that tests the function. This way each function used in TopoGen can be tested on its own. Each function has also been provided with documentation, which in an IDE can be called using the help function. Additionally, each step that is taken in a file is accompanied by comments. This approach was chosen specifically to make further development of TopoGen possible.

The most used Python library in TopoGen is the Pandas library. This is because using Pandas, boolean dataframes can be created to represent the different lists and UAMs. Table 3.1 provides an example of how a membership UAM between properties and ports would look like as a boolean dataframe, i.e. $\mu([\text{P-ID}] \times [\text{R-ID}])$. Besides storing the boolean variables, the UAMs rows and columns can be labelled with corresponding ULIDs. While technically not necessary, adding these labels makes verifying the calculations in TopoGen significantly easier. To create the ULIDs, the ULID-Python3 package is made use of, hosted on GitHub by Andrew Hawker [31]. Once the required lists and UAMs have been defined using the boolean dataframes, the dot function provided in Pandas allows boolean compositions to be performed, which, as will be shown in chapter 3, is one of the most essential operations executed in TopoGen.

	R-ID(1)	**R-ID(2)**	**R-ID(3)**
P-ID(1)	True	False	False
P-ID(2)	True	False	False
P-ID(3)	False	True	False
P-ID(4)	False	False	True

Table 3.1: Example of a boolean Pandas dataframe in order to represent a membership UAM between properties and ports, i.e. $\mu([\text{P-ID}] \times [\text{R-ID}])$.

3.2. POTENTIAL TOPOLOGIES WITHOUT CONSTRAINTS

When considering no constraints at all, then in PPLD, all ports can be connected to all other ports. Then with M: $M \in \mathbb{B}^{R_u \times R_y}$ (see section 3.1.2.5), as was explained in section 3.1.1, there exists a finite set of potential topologies $\mathcal{T}_p := \{M_1..M_{R_u R_y}\}$. The number of potential topologies (TPs) is expressed in (3.8), and represents the highest upper bound on the number of (feasible) topologies that exist given any set of components with R_u input ports and R_y output ports.

$$|\mathcal{T}_p| = 2^{R_u R_y} \tag{3.8}$$

This is the same expression that is already commonly mentioned in literature, and is used to indicate just how fast the number of TPs grows. (See chapter 2 for more on this.)

If (3.8) is applied to the 3.2BT-3.2EM test (see section 3.1.3), then $|\mathcal{T}_p| = 2^{12 \cdot 48} = 2^{576}$ TPs exist. This is far more than even the number of atoms in the observable universe [32], and it therefore does not make sense to "simply" verify all M matrices to see if they would produce also a feasible topology.

Table 3.2 shows an overview of the results for the 3.2BT-3.2EM test, which gets updated in each section. The computation time has been left out for the current case, as (3.8) is quick to compute.

	Base-2	**Base-10**	**Computation time [s]**
$\lvert\mathcal{T}_p\rvert$	2^{576}	2.47×10^{173}	-

Table 3.2: Results for 3.2BT-3.2EM test. (Table gets updated in the next sections.)

3.3. POTENTIAL TOPOLOGIES WITH BIJECTIVE CONNECTIONS

In this section the port-to-port constraint also used in [7] is applied as a first constraint, which acts on all components. The port-to-port constraint specifies that any port can only be connected to one other port.

3.3.1. WHY USE THE PORT-TO-PORT CONSTRAINT?

There are a few reasons why this constraint should be considered, and why the PLD and PPLD have been defined with this constraint already in mind. These are listed below:

1. **Energy conservation:** To maintain energy conservation, the output cannot be split into multiple outputs without also splitting the power. This needs to be taken into account somehow. The choice here was to let the components take care of splitting the power as required, and then provide multiple outputs. This way the ISM setup (see section 3.1.1) does not need to be modified. The port-to-port constraint can be applied then globally, while still allowing then any component to direct power to multiple other components.

2. **N-to-1 connection ambiguity:** The output of one component can be sent to the inputs of different components, i.e. 1-to-n connections. However the other way around, with n-to-1 connections, where multiple outputs are fed to one input, the question can be raised as to what exactly should be done with all the outputs feeding into one input. Should the sum of the outputs be considered? Or should the outputs considered differently? Again, it is best to define components in such a way that n outputs can only be fed to a component if it has n (compatible) inputs, and then let the model of the component specify how to handle all the different outputs that have been fed to it. Components would therefore be defined as MIMO systems, while one can continue using the ISM setup, and apply the port-to-port constraint.

3. **Minimum combinations for connections:** Consider a single reference port, in order to look into how many combinations h there exist of connecting it to r other ports when there are in total n ports, and one cannot connect to the same port multiple times. In [33] (3.9) is provided for this case. (See appendix A.2 for a full overview.)

$$h = \frac{n!}{r!(n-r)!} \tag{3.9}$$

Figure 3.11 provides the results obtained with (3.9) by plotting how the number of combinations grow when connecting to r other ports from a total of n other ports as the number n is increased. From (3.9) and Figure 3.11 it can be observed that when $r = 1$, i.e. a port can only connect to one other port, then the number of combinations grows linearly. As r is increased though, the growth in combinations increases significantly. This is yet another argument to not use CLD, where one would be exposed to situations where $r > 1$, and continue using PLD or PPLD, where one can enforce the port-to-port constraint such that $r = 1$.

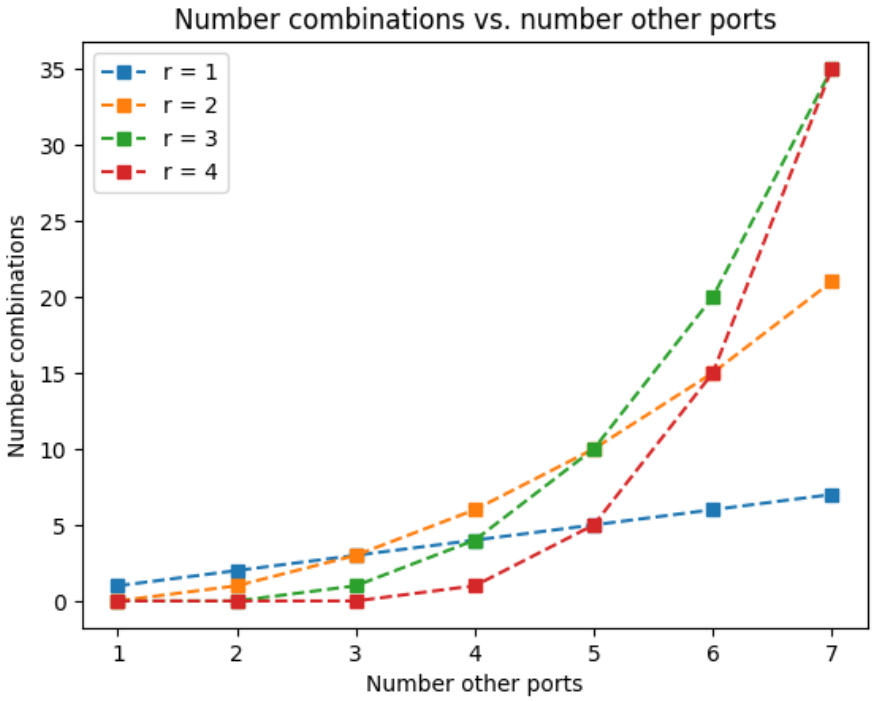


Figure 3.11: Plot that shows how the number of combinations grow when connecting to r other ports from a total of n other ports as the number n is increased.

3.3.2. IMPORTANT UAM PROPERTIES

When applying the port-to-port constraint, the M matrices that respect that constraint all have the same properties, namely:

- There is at most one 1 in each row of M, i.e. the sum of each row in M is at most one.
- There is at most one 1 in each column of M, i.e. the sum of each column in M is at most one.

This is because of how the M UAM is defined: As was explained in section 3.1.2.5, all ports referenced by the M UAM are unique, thus no selfloops are possible, and output ports can only connect to input ports. If therefore one also insists that a port can only connect to one other port, i.e. an output port can only connect to one other input port, then the conditions mentioned above are found.

Thus one can concentrate on all M UAMs that fulfil the conditions laid out in (3.10), and ignore all other M UAMs.

$$M \in \mathbb{B}^{R_u \times R_y} \tag{3.10}$$

s.t.

$$\left(\sum_{i=1}^{R_u} m_{i,j}\right) \leq 1 \quad \forall j \in [1..R_y]$$

$$\left(\sum_{j=1}^{R_y} m_{i,j}\right) \leq 1 \quad \forall i \in [1..R_u]$$

UAMs that fulfil the conditions mentioned in (3.10) are referred to here as bijective UAMs. Similarly, UAMs that only fulfil one of the two conditions are referred to as injective and surjective UAMs, where injective UAMs only meet the condition that the sum of each row is at most one, while surjective the condition that the sum of each column is at most one.

These definitions have been made with some inspiration from the definition of functions. (See appendix A.3 for how functions are defined.) The definitions do not match, but have been helpful. After all, any UAM can be seen as a mapping from the set of column indices C to the set of row indices R, i.e. a UAM is a function $f: C \mapsto R$. Thus for M, where the column indices are the output ports [R-ID|N-ID(y)] and the row indices are the input ports [R-ID|N-ID(u)], then f_M: [R-ID|N-ID(y)] $\mapsto$ [R-ID|N-ID(u)].

Besides defining UAMs as bijective, injective or surjective, a stricter definition has also been put together. Here the sums for each row or column (or both) should not be less or equal to one, but are equal to one. Thus a strict bijective UAM has the sum of all rows and columns equal to one.

As it turns out, all UAMs treated here meet one of these definitions. For example μ([P-ID] × [R-ID]) is a strict injective UAM, while μ([C-ID] × [H-ID]) is a strict surjective UAM. This also implies that the UAMs used here are very sparse. This inspires the question of how UAMs may be represented more data-efficient. This is addressed a bit further in the recommendations in chapter 4.

3.3.3. NUMBER OF POTENTIAL TOPOLOGIES WITH BIJECTIVE CONNECTIONS

In the previous section it was determined that for the port-to-port constraint to be respected, the M UAMs need to be bijective, i.e. the connections from port to port are bijective. Using (3.11) it can be explored how many bijective $M \in \mathbb{B}^{R_u \times R_y}$ UAMs there exist for R_u input ports and R_y output ports.

$$|\mathcal{T}_{p,b}| = \sum_{r=0}^{\min(R_u,R_y)} \left(\frac{R_u!}{r!(R_u-r)!}\right)\left(\frac{R_y!}{(R_y-r)!}\right) \tag{3.11}$$

Proof. Again the equations for the number of combinations or permutations from [33] and shown in Table A.1 are useful here:

First, it can be determined that if the port-to-port constraint is to be respected, then only a maximum of $\min(R_u, R_y)$ connections can be made.

Next, choose r connections. Then for those r connections, a combination of r input ports is chosen without repetition. In total, there will be $h_1 = \frac{R_u!}{r!(R_u-r)!}$ combinations. For each combination of input ports, one by one, extract r output ports from the set of output ports. This leads to a permutation of r output ports. In total, for each combination, there are $h_2 = \frac{R_y!}{(R_y-r)!}$ permutations. Thus when considering r connections, there will be $h_1 h_2$ options. Putting all of this together, the total number of bijective M UAMs, i.e the number of potential topologies with bijective connections (TPBs), can be expressed as shown in (3.11). ■

Using (3.11), the number of potential topologies with bijective connections (TPBs) for the 3.2BT-3.2EM test can be calculated. The results are shown in Table 3.3. Again the computation time has been left out, since (3.11) is fast to compute.

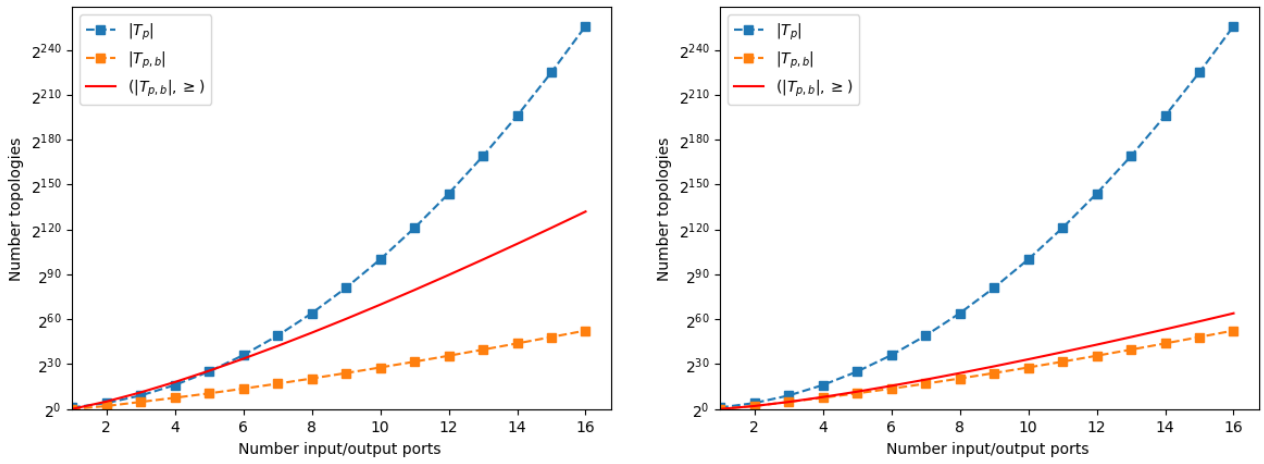
	Base-2	Base-10	Computation time [s]
$\lvert\mathcal{T}_p\rvert$	2^{576}	2.47×10^{173}	-
$\lvert\mathcal{T}_{p,b}\rvert$	2^{65}	4.61×10^{19}	-

Table 3.3: Results for 3.2BT-3.2EM test. (Table gets updated in the next sections.)

It is clear from the results shown in Table 3.3 that only applying the port-to-port constraint already reduces the majority of the available TPs.

A more general comparison can be made between the case of no constraints versus the case where the port-to-port constraint is applied. To make this comparison, choose M to be square, so that the number of input ports is the same as the output ports, i.e. $R_u = R_y =: R$.

In Figure 3.12 it is shown how the number of TPs increases with R ports. The blue line shows how the number of TPs without constraints increases with R, while the yellow shows it for the TPBs. It is immediately clear that the growth of the latter case is significantly shallower compared to the unconstrained case.



(a) The number of TPBs is bounded by $(|\mathcal{T}_{p,b}|, \geq) = R^{2R+1}$ (red line).

(b) The number of TPBs is bounded by $(|\mathcal{T}_{p,b}|, \geq) = R^R$ (red line).

Figure 3.12: Number of TPs (in blue) versus TPBs (in yellow). Additionally two upper bounds are shown (red line).

To express how much shallower the growth is for the TPBs, an upper bound is expressed using Sterling's approximation. (See appendix A.4 for a brief description of Sterling's approximation.) With this approximation it can be shown that an upper bound for $|\mathcal{T}_{p,b}|$ is $(|\mathcal{T}_{p,b}|, \geq) = R^{2R+1}$, also visualised in Figure 3.12a. See the proof below for the derivation of this upper bound.

Proof. Setting $R_u = R_y =: R$ implies that (3.11) is expressed as:

$$|\mathcal{T}_{p,b}| = \sum_{r=0}^{R} \left(\frac{R!^2}{r!(R-r)!^2} \right) \tag{3.12}$$

A simple upper bound for (3.12) is $|\mathcal{T}_{p,b}| \leq R(R!)^2$. Applying $\log_2$, the following derivation can be

made:

$$\begin{aligned}\log_2(|\mathcal{T}_{p,b}|) &\leq \log_2\left(R(R!)^2\right)\\ &= \log_2(R) + 2\log_2(R!)\end{aligned}$$

Then, using Stirling's approximation, where $n\ln(n) \geq \ln(n!)$, an upper bound to the last expression can be formed:

$$\begin{aligned}\log_2(|\mathcal{T}_{p,b}|) &\leq \log_2(R) + 2\log_2(R!)\\ &\leq \log_2(R) + 2R\log_2(R)\\ &= (1+2R)\log_2(R)\end{aligned} \tag{3.13}$$

Thus R^{2R+1} is an upper bound to $|\mathcal{T}_{p,b}|$, i.e. $(|\mathcal{T}_{p,b}|, \geq) = R^{2R+1}$. ■

When the upper bound $(|\mathcal{T}_{p,b}|, \geq) = R^{2R+1}$ is simplified to $(|\mathcal{T}_{p,b}|, \geq) = R^R$, as shown in Figure 3.12b, it seems to be an even closer upper bound to $|\mathcal{T}_{p,b}|$. A proof remains to be given for this upper bound, but it still seems to be safe to conclude that while $|\mathcal{T}_p| = 2^{R^2}$, $|\mathcal{T}_{p,b}| \leq R^R$. (Or in logarithmic form, while $\log_2\left(|\mathcal{T}_p|\right) = R^2$, $\log_2\left(|\mathcal{T}_{p,b}|\right) \leq R\log_2(R)$.)

Considering that (3.11) only takes into account the number of input and output ports in a set of components $\mathcal{S}_c$, as long as a port based description is used like it is done in PLD or PPLD, (3.11) is a significantly better upper bound for any set of components $\mathcal{S}_c$ compared to the much more often mentioned in literature upper bound of (3.8).

3.3.4. CONCLUSION

In this section the port-to-port constraint was applied, and it was studied what implications this constraint has on the M UAMs and the number of TPs that remain.

The work shown here is a significant contribution to SRQ-2 (see section 2.3) and to the general problem of topology generation, since none of the material shown here is present in the literature that has been studied. The reason that this has not been mentioned so far is most likely because the use of matrices for topology generation has not been applied yet. Without the use of matrices, observing the UAM properties that exist or deriving (3.11) is significantly harder. Yet having done so, a significantly better upper bound has been found for any set of components $\mathcal{S}_c$ when using a port-based description for the components.

Additionally, writing the port-to-port constraint using UAMs means that the constraint is expressed as a global constraint. This is a useful contribution to SRQ-1, even though it is likely already done in a similar fashion by the current state of the art.

In the next section the compatibility between ports will be considered, to see what impact this second constraint has on reducing the number of TPs.

3.4. POTENTIAL TOPOLOGIES WITH BIJECTIVE COMPATIBLE CONNECTIONS

In the previous section the port-to-port constraint, also used in [7], was applied as a first constraint, which acted on all components. The port-to-port constraint specified that any port can only be connected to one other port.

The natural continuation from this is to insist furthermore that a port can only connect to another port if it is compatible. This is what will be treated in this section.

3.4.1. DEFINING THE COMPATIBILITY MATRIX

To define compatibility, each element in PPLD is considered.

MATCHING UNIT & DOMAINS TYPES

To start with it is insisted upon that properties can only connect to other properties if they share the same unit and domain types. To express this constraint, the boolean composition described in section 3.1.2.4 is made use of. It turns out that if for example the membership UAM between properties and unit types, i.e. $\mu([\text{P-ID}] \times [\text{V-ID}])$, is composed with its own transpose, what is given is a membership UAM that specifies which properties have matching unit types, i.e. $\mu([\text{P-ID}] \times [\text{P-ID}]||[\text{V-ID}])$. Equation (3.14) presents this formulation, while (3.15) presents which properties have matching domain types.

$$\mu([\text{P-ID}] \times [\text{V-ID}]) \circ \mu([\text{P-ID}] \times [\text{V-ID}])^T = \mu([\text{P-ID}] \times [\text{P-ID}]||[\text{V-ID}]) \tag{3.14}$$

$$\mu([\text{P-ID}] \times [\text{D-ID}]) \circ \mu([\text{P-ID}] \times [\text{D-ID}])^T = \mu([\text{P-ID}] \times [\text{P-ID}]||[\text{D-ID}]) \tag{3.15}$$

SWITCHING BETWEEN NODE TYPES

Next it is defined that input properties can only connect to output properties and vice versa. The condition is expressed similar to how (3.14) and (3.15) were defined, but a negation is used to select all properties that do not have a matching node type. This is shown in (3.16).

$$\neg\left(\mu([\text{P-ID}] \times [\text{N-ID}]) \circ \mu([\text{P-ID}] \times [\text{N-ID}])^T\right) = \neg\mu([\text{P-ID}] \times [\text{P-ID}]||[\text{N-ID}]) \tag{3.16}$$

ONLY CONNECT TO OTHER PORTS

Properties of one port shall only connect to properties of a different port. This is expressed in (3.17).

$$\neg\left(\mu([\text{P-ID}] \times [\text{R-ID}]) \circ \mu([\text{P-ID}] \times [\text{R-ID}])^T\right) = \neg\mu([\text{P-ID}] \times [\text{P-ID}]||[\text{R-ID}]) \tag{3.17}$$

ONLY CONNECT TO OTHER INSTANCES

Properties of one instance shall only connect to properties of a different instance. This is expressed in (3.18).

$$\neg\left(\mu([\text{P-ID}] \times [\text{H-ID}]) \circ \mu([\text{P-ID}] \times [\text{H-ID}])^T\right) = \neg\mu([\text{P-ID}] \times [\text{P-ID}]||[\text{H-ID}]) \tag{3.18}$$

ONLY CONNECT TO OTHER COMPONENTS

Properties of one component shall only connect to properties of a different component. This is expressed in (3.19).

$$\neg\left(\mu([\text{P-ID}] \times [\text{C-ID}]) \circ \mu([\text{P-ID}] \times [\text{C-ID}])^T\right) = \neg\mu([\text{P-ID}] \times [\text{P-ID}]||[\text{C-ID}]) \tag{3.19}$$

ONLY CONNECT TO OTHER COMPONENT TYPES

Finally, it is also assumed that properties of one component type shall only connect to properties of a different component type. This is expressed in (3.20). While there are assemblies that connect component types in series, for example batteries, when connected in series, here they will be regarded as one single component. This is to avoid having many topologies whose difference is only how many for example batteries have been connected in series.

$$\neg\left(\mu\left([\text{P-ID}]\times[\text{T-ID}]\right)\circ\mu\left([\text{P-ID}]\times[\text{T-ID}]\right)^{T}\right)=\neg\mu\left([\text{P-ID}]\times[\text{P-ID}]||[\text{T-ID}]\right) \tag{3.20}$$

Combining multiple components into one is also already a strategy implemented in [11] as a way to reduce the number of components in a set of components $\mathcal{S}_c$. In [11] components that are known to be connected together, such as a battery and an electric motor, are connected apriori, forming a single component and replacing the previous two.

This strategy can also be used to provide more flexibility in how each component is defined. For example usually, to connect a battery to an electric motor, an inverter is placed in between to control the voltage supplied to the motor. However here the battery is assumed to already have the inverter, and thus a battery+inverter can be modelled as a single battery component that can connect directly to an electric motor.

Whatever the reason may be to join components together, once they have been, through (3.20) it is maintained that components then only connect to components of a different component type.

PUTTING IT ALL TOGETHER

The expressions from (3.14) to (3.20) can then be combined using the AND operation, as is shown in (3.21).

$$\begin{aligned}\zeta\left([\text{P-ID}]\times[\text{P-ID}]\right):=\ &\mu\left([\text{P-ID}]\times[\text{P-ID}]||[\text{V-ID}]\right)\ \wedge\\ &\mu\left([\text{P-ID}]\times[\text{P-ID}]||[\text{D-ID}]\right)\ \wedge\\ &\neg\mu\left([\text{P-ID}]\times[\text{P-ID}]||[\text{N-ID}]\right)\ \wedge\\ &\neg\mu\left([\text{P-ID}]\times[\text{P-ID}]||[\text{R-ID}]\right)\ \wedge\\ &\neg\mu\left([\text{P-ID}]\times[\text{P-ID}]||[\text{H-ID}]\right)\ \wedge\\ &\neg\mu\left([\text{P-ID}]\times[\text{P-ID}]||[\text{C-ID}]\right)\ \wedge\\ &\neg\mu\left([\text{P-ID}]\times[\text{P-ID}]||[\text{T-ID}]\right)\end{aligned} \tag{3.21}$$

The expression $\zeta([\text{P-ID}]\times[\text{P-ID}])$ therefore is a compatibility UAM that specifies what properties are compatible with each other, taking into account the entire PPLD.

However, since the connection UAM M specifies the connections made between ports, what is needed is a compatibility UAM that specifies what ports are compatible with each other, i.e. $\zeta([\text{R-ID}]\times[\text{R-ID}])$. The operations shown in (3.22) can be executed to obtain $\zeta([\text{R-ID}]\times[\text{R-ID}])$. Similar operations can be executed to obtain also the compatibility between instances, components and component types.

$$\begin{aligned}\zeta\left([\text{P-ID}]\times[\text{P-ID}]\right)\circ\mu\left([\text{P-ID}]\times[\text{R-ID}]\right)&=\zeta\left([\text{P-ID}]\times[\text{R-ID}]\right)\\ \zeta\left([\text{P-ID}]\times[\text{R-ID}]\right)^{T}\circ\mu\left([\text{P-ID}]\times[\text{R-ID}]\right)&=\zeta\left([\text{R-ID}]\times[\text{R-ID}]\right)\end{aligned} \tag{3.22}$$

Nevertheless, the operation in (3.22) is excessive. As it has already been established, the only connections that are going to be manipulated in M are connections from output ports to input ports. Thus not all of $\zeta([\text{R-ID}] \times [\text{R-ID}])$ is needed, but only $\zeta_r([\text{R-ID}|\text{N-ID}(u)] \times [\text{R-ID}|\text{N-ID}(y)])$, i.e. the compatibility between output and input ports.

While it is possible to first compute the entire port compatibility matrix $\zeta([\text{R-ID}] \times [\text{R-ID}])$, and then downsize it to $\zeta_r([\text{R-ID}|\text{N-ID}(u)] \times [\text{R-ID}|\text{N-ID}(y)])$ using $\mu_r([\text{R-ID}|\text{N-ID}(u)])$ and $\mu_r([\text{R-ID}|\text{N-ID}(y)])$, such approach means that first a big matrix was computed, only to erase large parts of it again. A more direct and data-efficient derivation is to directly work with inputs and outputs separately, as is shown in the derivations of (3.23) shown below:

$$\begin{aligned}
&\zeta([\text{P-ID}] \times [\text{P-ID}]) \circ \mu_r([\text{P-ID}] \times [\text{R-ID}|\text{N-ID(u)}]) = \zeta_r([\text{P-ID}] \times [\text{R-ID}|\text{N-ID}(u)]) \\
&\zeta([\text{P-ID}] \times [\text{P-ID}]) \circ \mu_r\left([\text{P-ID}] \times [\text{R-ID}|\text{N-ID(y)}]\right) = \zeta_r\left([\text{P-ID}] \times [\text{R-ID}|\text{N-ID}(y)]\right)
\end{aligned}$$

$$\begin{aligned}
\zeta_r([\text{P-ID}] \times [\text{R-ID}|\text{N-ID}(u)])^T \circ \zeta([\text{P-ID}] \times [\text{P-ID}]) \circ \zeta_r\left([\text{P-ID}] \times [\text{R-ID}|\text{N-ID}(y)]\right) = \\
= \zeta_r\left([\text{R-ID}|\text{N-ID}(u)] \times [\text{R-ID}|\text{N-ID}(y)]\right) \qquad (3.23)
\end{aligned}$$

That the approach shown in (3.23) is indeed more efficient compared to the approach shown in (3.22) can be tested by computing the compatibility UAM $\zeta_r([\text{R-ID}|\text{N-ID}(u)] \times [\text{R-ID}|\text{N-ID}(y)])$ several times for the 3.2BT-3.2EM test and computing the average computation time in seconds. The results that were obtained are shown below, where the approach from (3.22) is referred to as 'Method 1', while 'Method 2' refers to the approach shown in (3.23). The results confirm that the approach with (3.23) is indeed more efficient.

```
1    Method 1 - 100 runs average: 0.002049839000974316
2    Method 2 - 100 runs average: 0.0016445649988600054
3    Method 1 - 1000 runs average: 0.0018061095999437385
4    Method 2 - 1000 runs average: 0.0015784566993825137
```

3.4.2. NUMBER OF POTENTIAL TOPOLOGIES WITH BIJECTIVE COMPATIBLE CONNECTIONS

Having computed the compatibility UAM between input and output ports, i.e. $\zeta_r([\text{R-ID}|\text{N-ID}(u)] \times [\text{R-ID}|\text{N-ID}(y)]) \in \mathbb{B}^{R_u \times R_y}$, now the number of potential topologies with both bijective and compatible connections (TPBCs) can be computed. Now the different $M \in \mathbb{B}^{R_u \times R_y}$ UAMs need to not just fulfil the conditions in (3.10) from section 3.3, but also can only contain 1's where there are 1's in $\zeta_r([\text{R-ID}|\text{N-ID}(u)] \times [\text{R-ID}|\text{N-ID}(y)])$, as is summarised in (3.24).

$$M \in \mathbb{B}^{R_u \times R_y} \tag{3.24}$$

s.t.

$$\left(\sum_{i=1}^{R_u} m_{i,j}\right) \leq 1 \quad \forall j \in [1..R_y]$$

$$\left(\sum_{j=1}^{R_y} m_{i,j}\right) \leq 1 \quad \forall i \in [1..R_u]$$

$$z_{i,j} = 0 : \; z_{i,j} \in \zeta_r([\text{R-ID}|\text{N-ID}(u)] \times [\text{R-ID}|\text{N-ID}(y)]) \quad \Longrightarrow \quad m_{i,j} = 0$$

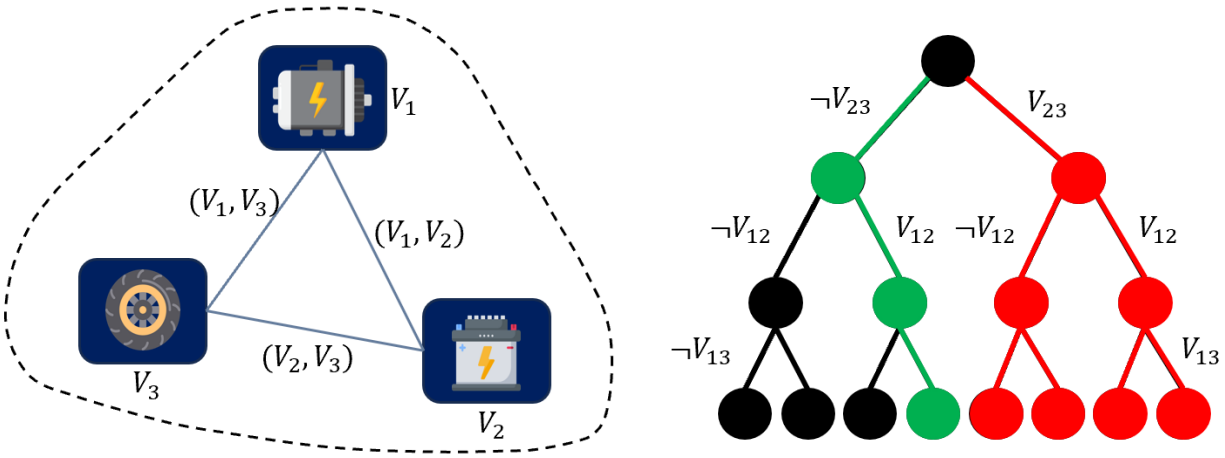
Unlike TPBs, where (3.11) was found to calculate the number of TPBs directly, no equivalent equation has been found that also takes into account port compatibility. Instead, to find the number of TPBCs, one must find the actual topologies that meet these conditions, i.e. find all M UAMs that satisfy the conditions mentioned in (3.24).

3.4.2.1. SAT SOLVERS TO FIND POTENTIAL TOPOLOGIES

At this point, literature would suggest to use the CSP solver, as explained in chapter 2. Unfortunately, in [8, 5], it is not recognised that the CSP can actually be simplified to a Satisfiability (SAT) problem. In [7] the use of SAT solvers is suggested for faster computations, but not explained either why it is possible to use SAT solvers for topology generation. In short, this can be done, because all variables considered in the current topology generation problem are boolean, i.e. 1 or 0. As is explained in [34], a lot of research effort has gone into SAT solvers. In [35] the techniques behind SAT solvers are explained in more detail. Given that the CSP problem behind topology generation can be simplified to a SAT problem, and that using SAT solvers would be using the current state of the art to tackle topology generation, in here SAT solvers were considered directly.

The basis of a SAT solver is a boolean search tree. In Figure 3.13 an example is shown, where the example set of components from chapter 2 is used again and visualised in Figure 3.13a. In Figure 3.13b the boolean search tree is shown, where at each level it is decided upon whether a connection is placed or not, forming in total eight branches that correspond to the eight TPs without constraints. The SAT solver explores the branches to find which branch does not violate any constraints. When one of the branches does violate a constraint, the SAT solver attempts to use that information to narrow down which branches are still worth exploring. For example, in chapter 2 one of the constraints that were placed was that the connection (V_2, V_3) was not compatible, i.e. $(V_2, V_3) = 0$. Thus, through constraint propagation, the SAT solver determines that the entire right half of the search tree does not need to be explored. A few more searches, and the SAT solver might find the solution that is shown as the green branch in Figure 3.13b.

Constraint propagation is only one of the techniques that is used by SAT solvers to narrow down the branches that need to be explored. Both [34] and [35] describe several other techniques, explaining then that the current state-of-the-art SAT solver will use the Conflict Driven Clause Learning (CDCL) algorithm to solve SAT problems as fast as possible. As also explained by both sources, SAT solvers can still be made to perform faster by using random restarts and machine learning, but these boosts are probabilistic, i.e. they may not work out. Thus a SAT solver should be chosen that, at the very least, makes use of the CDCL algorithm. In [36] a list of SAT solvers that make use of the CDCL algorithm is presented:



(a) Example set of components.

(b) Search tree used by SAT solver.

Figure 3.13: Search tree that is used by the SAT solver to find what connections to place for the example set of components considered in chapter 2.

- MiniSAT
- Zchaff SAT
- Z3
- Glucose
- ManySAT

The Z3 SAT solver seems to be a popular choice for people using Python, and sufficient information is available online that describes how to use it. For example in [37] it is described how to use the Z3 SAT solver to solve a classical SAT problem. The Z3 SAT solver was developed by Microsoft Research, and is available on GitHub (see [38]) with an MIT license, thus open to be used by any person. In their repository it is again confirmed that "the engine used by default is based on the CDCL architecture".

Thus the Z3 SAT solver is chosen to find all M UAMs that meet the conditions laid out in (3.24). To describe the row and column sum constraints in (3.24), pseudo-boolean constraints are used as suggested by [39]. These rewrite the boolean expression into linear equations, and Z3 provides the `PbLe`, `PbEq` and `PbGe` functions to specify such constraints. (See a brief description on pseudo-boolean constraints in appendix A.5.) The code for this SAT solver is available in the TopoGen repository (see section 3.1.4) at `~!tests/compare-SAT-vs-CP/compute_bijective_topologies_SAT.py`.

When running the SAT solver for the 3.2BT-3.2EM test, it specifies that there are $|\mathcal{T}_{p,bc}| = 13'326$ TPBCs. The result is presented in Table 3.4.

Alarmingly it took the SAT solver close to two minutes to obtain this answer. When reviewing the UAMs it is found that $\zeta_r([\text{R-ID}|\text{N-ID}(u)] \times [\text{R-ID}|\text{N-ID}(y)])$ reduces the connections described by the $M \in \mathbb{B}^{12\times 48}$ UAM to a UAM of only compatible connections of $M_c \in \mathbb{B}^{6\times 6}$. In other words, it takes a state-of-the-art SAT solver close to two minutes to process a six-by-six UAM. This was considered to be highly suspicious, since a significantly faster computation time was expected

	Base-2	Base-10	Computation time [s]
$\lvert\mathcal{T}_p\rvert$	2^{576}	2.47×10^{173}	-
$\lvert\mathcal{T}_{p,b}\rvert$	2^{65}	4.61×10^{19}	-
$\lvert\mathcal{T}_{p,bc}\rvert$	2^{14}	13.3×10^{3}	113.28

Table 3.4: Results for 3.2BT-3.2EM test. (Table gets updated in the next sections.)

for such a size.

To explore a bit further, the SAT solver was tested on further square UAMs from size 2 to 6, where as the worst case scenario it was assumed that all connections on these UAMs were compatible. This would be the least constrained cases for which the SAT solver would require the most amount of computation time [7]. (As such the problem is simplified to finding the TPBs, but unlike section 3.3 where only the number of TPs was calculated, here the topologies are retrieved.) The results are presented in Figure 3.14, where it can be seen that the computation time grows quadratically on the log-2 scale, and already with a five-by-five UAM, the required computation time is close to eight seconds.

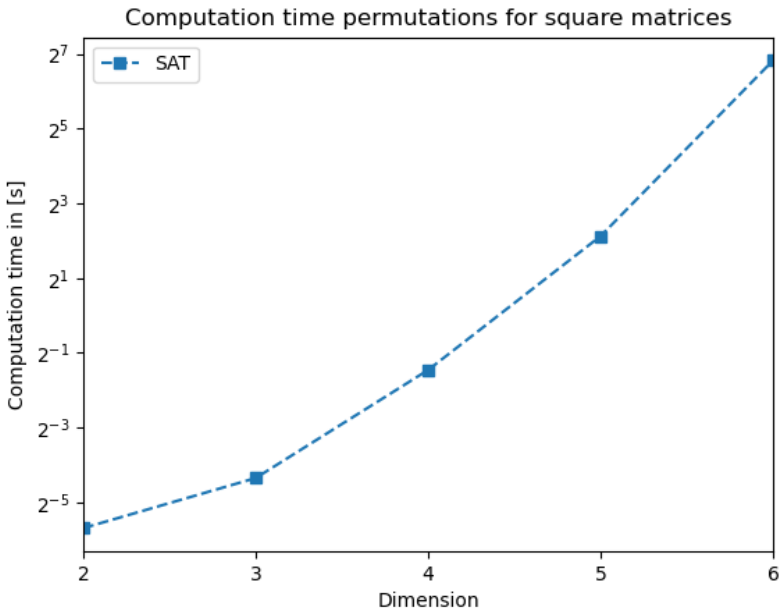


Figure 3.14: Computation time required by SAT solver to find all TPBCs versus the dimension of the tested square UAM.

Attempting to improve on this, the advice laid out in [7] was reviewed. There it is described that CSP solvers (and thus also SAT solvers) work more efficient the more constrained the CSP/SAT problem is. This inspired to look into whether the M UAM could be constrained in more ways. These would have to be redundant constraints, that are merely there so that the SAT solver can eliminate certain branches faster, but would have been eliminated by the constraints mentioned in (3.24) eventually anyways.

Indeed such a redundant constraint has been found. After all, given R_u input ports and R_y output ports, as was explained in section 3.3 it is known that only a maximum of $\min(R_u, R_y)$ connections can be made. That also means that the SAT problem described in (3.24) can be broken down into multiple SAT problems, where in each only topologies with r connections are explored. Thus with $M_c \in \mathbb{B}^{6\times6}$, the SAT problem from (3.24) is broken down to 6 SAT problems, with each problem exploring $r \in [1..6]$ connections. This strategy is referred to as the multi-SAT strategy, and now the number of TPBCs is obtained within 34.8 s instead of the 113.3 s from before. (See the updated table of results, Table 3.5.) When repeating the test on different square UAMs, the yellow curve in Figure 3.15 was obtained. While for UAMs up to dimension four the regular SAT solver is faster, the growth in computation time for the multi-SAT is significantly shallower, meaning that for UAMs of dimension five and upward, the multi-SAT approach is significantly faster. This still means though that it takes a state-of-the-art SAT solver considerable time to find all topologies even for a six-by-six UAM.

	Base-2	**Base-10**	**Computation time [s]**
$\lvert\mathcal{T}_p\rvert$	2^{576}	2.47×10^{173}	-
$\lvert\mathcal{T}_{p,b}\rvert$	2^{65}	4.61×10^{19}	-
$\lvert\mathcal{T}_{p,bc}\rvert$	2^{14}	13.3×10^{3}	34.8

Table 3.5: Results for 3.2BT-3.2EM test. (Table gets updated in the next sections.)

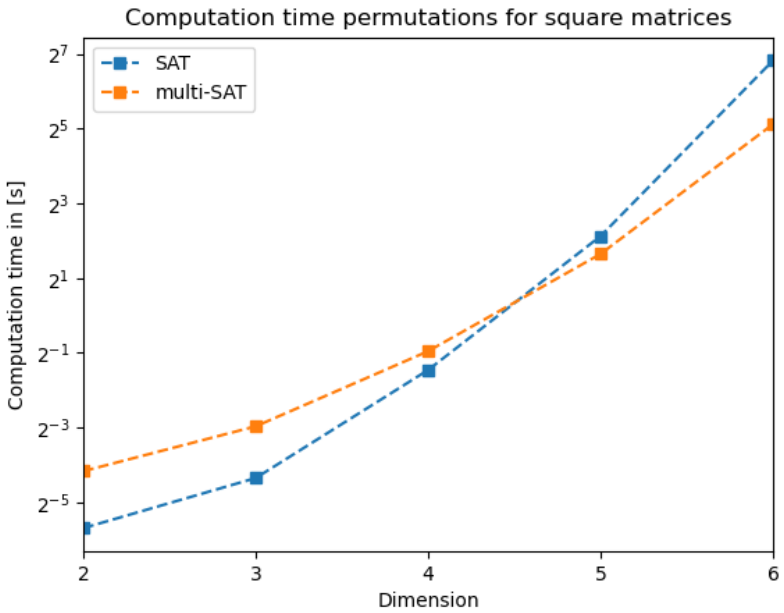
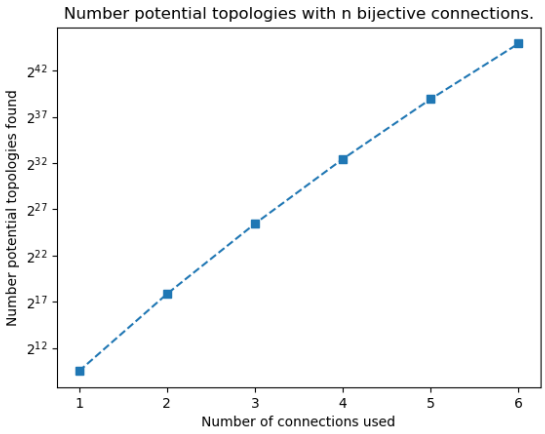


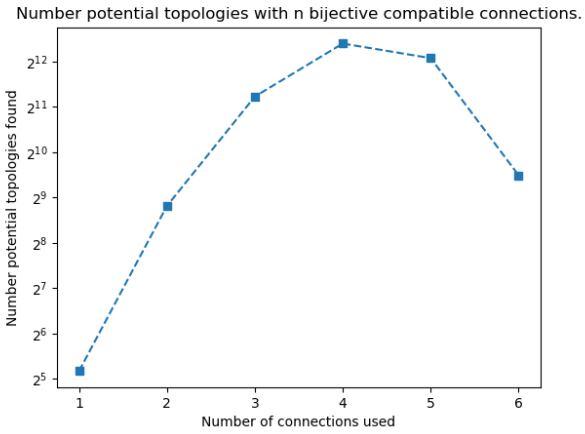
Figure 3.15: Computation time required by SAT and multi-SAT solver to find all TPBCs versus the dimension of the tested square UAM.

Still, an added benefit of breaking down the SAT problem from (3.24) into problems that only consider r connections is that instead of only providing one total number of TPBCs, i.e. $|\mathcal{T}_{p,bc}|$, now the number of TPBCs with r connections can be given. The results for the 3.2BT-3.2EM test

are shown in Figure 3.16, where in Figure 3.16a the number of TPBs with r connections is shown, whereas in Figure 3.16b also compatibility is taken into account. Not only are there significantly fewer TPBCs than TPBs, but also the number of TPBCs with r connections first increases as r is increased, but then achieves a peak at $r = 4$. (Unlike the case with bijective connections only, which keep increasing as r is increased with the same growth that was already analysed in section 3.3.) Beyond $r = 4$, as r is increased further to the maximum number of connections that can be considered, $r = 6$, the number of TPBCs decreases again. This is a result that matches with the results found in [8], where there too, in Figure 12 of [8], a similar trend was found for their set of components. Thus the results obtained here are similar to what is present in literature, which adds to the validity of the results obtained here.



(a) Number of TPBs with r connections.



(b) Number of TPBCs with r connections.

Figure 3.16: Number of topologies with r connections versus the number of connections r used.

3.4.2.2. CONSTRAINED PERMUTATION GENERATOR TO FIND POTENTIAL TOPOLOGIES

Given that even the multi-SAT approach described in section 3.4.2.1 is slow in retrieving the TPBCs, an effort was taken to look whether next to SAT solvers there is any other algorithm that could be used to solve this SAT problem.

Inspiration was taken from the approach that was used to put together (3.11), which was used to calculate $|\mathcal{T}_{p,b}|$. (See the proof for (3.11) in section 3.3.) There a combination of r input ports was chosen, and then one by one, from the set of output ports, r output ports were extracted. Here now, what is needed is that depending to which input port one is connecting to, an output from only a compatible set of output ports is extracted from.

Consider as an example a compatibility UAM $Z \in \mathbb{B}^{3\times 3}$ shown in (3.25). It states that the input port in the first row can be connected by any of the output ports. However the next two input ports can only be connected by the output ports present in column two and three.

$$Z = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \tag{3.25}$$

The UAM Z can be converted into a list of lists, which stores the column indices, i.e. the indices of the output ports, where a 1 is located. This conversion is shown in (3.26), and the list of lists is referred here to as an Unweighted Adjacency List (UAL).

$$Z_{\text{UAM}} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \rightsquigarrow Z_{\text{UAL}} = [[1,2,3],[2,3],[2,3]] \tag{3.26}$$

Thus to form from Z TPBCs, permutations from Z_{UAL} need to be formed where the first output port is from the first list, the second output port is from the second list, and so on. This is referred to as developing constrained permutations. In the example shown here, for $r = 3$ connections, i.e. all lists in Z_{UAL} are used, only two constrained permutations exist, $[1,2,3]$ and $[1,3,2]$. Thus for Z there are only two TPBCs with $r = 3$ connections. To find all TPBCs, for some compatibility UAM $Z \in \mathbb{B}^{p\times q}$, the constrained permutations for all combinations of input ports with $r \in [1..\min(p,q)]$ connections need to be found. This is similar to what was done with the multi-SAT solver.

Luckily enough, an algorithm that would develop these constrained permutations, i.e. a Constrained Permutation Generator (CPG), was found online. The script was written by Tim Peters and is available on Stack-overflow at [40]. The script has been copied and modified with more readable variable names and comments that describe each step in the algorithm. Additionally, the rows and columns of the compatibility UAL, which is the input for the CPG, are ordered in such a way that the shorter lists appear first. This speeds up the CPG even further, since permutations that have values not in the UAL are found faster and dismissed. (See proof below.) The adjusted version is available at the TopoGen repository at `~!tests/compare-SAT-vs-CP/generate_constrained_permutations.py`. The script was also tested on multiple random UAMs to confirm that the results always match the results produced with the SAT and multi-SAT solver. Additionally, because the CPG needs to be run on all combinations of input ports with r connections for all r, the number of TPBCs with r connections can be given, just like it was done with the multi-SAT solver and shown in Figure 3.16b.

Proof. The permutations are developed by extracting the first output port from the first list in the UAL, then the second output port from the second list in the UAL, and so on. If the permutation that is being developed arrives at the n-th list and requires the n-th value, but this value is not present in the n-th list, the permutation can be dismissed. The sooner this happens, the less the permutation needs to be developed/tested. For example, if the first list only has one value, then only permutations with the first value as the value in the first list need to be tested. ■

The test on different square UAMs that was done in section 3.4.2.1 was run again using the CPG. The results are shown as the green line in Figure 3.17. It is immediately clear that the CPG significantly outperforms both the SAT and the multi-SAT solver. With a UAM of size two, the CPG is about $2^{12-5} = 2^7 = 128$ times faster. However the rate at which the computation time increases is also lower for the CPG, even lower than the rate of the multi-SAT solver. That means that when dealing with a UAM of size six, the CPG now outperforms the multi-SAT solver by about $2^{4+8} = 2^{12} = 4'096$ times. When running the CPG to find all TPBCs for the 3.2BT-3.2EM test, the CPG obtains the result in 12.1×10^{-3} s in comparison to the 34.8 s it took for the multi-SAT to find the result. This is shown in the updated table Table 3.6. Given how much the CPG outperforms the SAT solvers used here, it is integrated into the main TopoGen program at `generate_topologies/generate_constrained_permutations.py`.

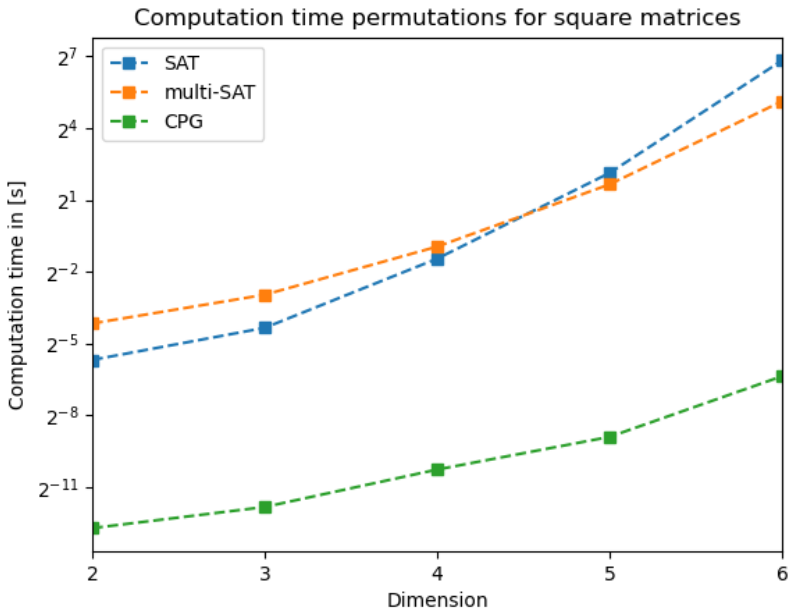


Figure 3.17: Computation time required by SAT, multi-SAT and CPG to find all TPBCs versus the dimension of the tested square UAM.

The hypothesis as to why the CPG outperforms the SAT solvers is as follows: Unlike the boolean search tree that is used by the SAT solvers, the CPG develops a permutation search tree. While the bijective constraints constrain the SAT solver to only produce permutations, the SAT solver only develops these permutations as a result of testing other cases and those cases failing as they do not meet the constraints. The CPG meanwhile directly works with permutations from the start. Additionally in the CPG search tree only has a length of p for a compatibility UAM $Z \in \mathbb{B}^{p \times q}$. The length of the branches in the SAT solver meanwhile are of length pq. The shorter branches might

	Base-2	Base-10	Computation time [s]
$\lvert\mathcal{T}_p\rvert$	2^{576}	2.47×10^{173}	-
$\lvert\mathcal{T}_{p,b}\rvert$	2^{65}	4.61×10^{19}	-
$\lvert\mathcal{T}_{p,bc}\rvert$	2^{14}	13.3×10^{3}	0.012

Table 3.6: Results for 3.2BT-3.2EM test. (Table gets updated in the next sections.)

lead to further increases in speed for the CPG.

Whatever the reason may be as to why the CPG outperforms the SAT solvers, it does not need to imply that SAT solvers are ill-suited for topology generation. It may well be that the SAT solvers require more redundant constraints before they can perform at the same rate as a CPG. Finding this out however, as well as explaining why the CPG outperforms the SAT solvers, is beyond the scope of this thesis, and needs to be dealt with in future research.

3.4.3. CONCLUSION

In this section it was shown how the compatibility constraint can be expressed as a compatibility UAM. Thanks to boolean composition and using membership UAMs, this too is a global constraint that can be applied to any set of components $\mathcal{S}_c$. Thus this approach is a major contribution to SRQ-1. (See section 2.3.)

Then SAT solvers were used to compute the number of TPBCs. Again, these computations can be done for any set of components $\mathcal{S}_c$, and are a closer upper bound than the bound found in section 3.3. This is a useful contribution to SRQ-2. Not only that, but it is shown how the results resemble those obtained in the literature.

Unfortunately it was shown how SAT solvers, at least with the setup that was used here, are very slow to retrieve all TPBCs. Instead, an alternative method that relied on combinations and permutations was developed, the CPG method, which provided results significantly faster.

All in all, this section provides again a lot of useful insight for topology generation in general. Now that both the port-to-port and the compatibility constraint have been set in place, the setup is complete to tackle the MRQ. This involves computing the feasible topologies (TF) while preventing isomorphic topologies from emerging, which will be done in the next section.

3.5. FEASIBLE TOPOLOGIES WITHOUT ISOMORPHIC TOPOLOGIES

So far only the set of components $\mathcal{S}_c$ has been considered. Applying both the port-to-port constraint (see section 3.3) and the compatibility constraint (see section 3.4), the number of TPs could be reduced significantly. In this section though, in order to obtain the feasible topologies (TFs), the set of requirements $\mathcal{S}_r$ will be made use of. Here, as was explained previously, only a single output requirement will be considered, as this is enough to answer the MRQ. In other words, $\mathcal{S}_r = \{R_o\}$, where R_o is the output requirement. The main challenge lies in creating the TFs, while the isomorphic topologies are avoided.

3.5.1. METHOD TO AVOID ISOMORPHIC TOPOLOGIES

As was explained in chapter 2, currently no method exists that can avoid isomorphic topologies. Here a method is proposed to tackle that problem.

As it is understood, the reason that isomorphic topologies are generated is because the instances of a component cannot be distinguished. The method here attempts to correct that.

This is done by enforcing an order on the instances. As the TFs are developed, the instances are used one by one in that order. A practical example of this is a warehouse, where one might store multiple instances of a battery component. It makes sense to then supply the oldest instances first, and keep supplying then ever newer instances.

Here the instances can already be ordered thanks to the use of ULIDs. (See section 3.1.2.2.) Consider the membership UAM $\mu([\text{C-ID}] \times [\text{H-ID}])$, where both the components and instances are ordered by their corresponding ULIDs[3]. An example of such a UAM is shown in Table 3.7, where each of the three components has two instances available. Then choosing the first instance of any component involves choosing the left-most instance on $\mu([\text{C-ID}] \times [\text{H-ID}])$. In Table 3.7 the first instances are highlighted. Once the first instances have been connected, the next instances are used when needed. A limit has been reached when no more instances for a component are available.

	H(1)	H(2)	H(3)	H(4)	H(5)	H(6)
C(1)	0	0	1	1	0	0
C(2)	1	1	0	0	0	0
C(3)	0	0	0	0	1	1

Table 3.7: Example of the membership UAM $\mu([\text{C-ID}] \times [\text{H-ID}])$, with the first instances of each component highlighted.

[3] Actually simply by creating a membership UAM, an order is created. However one must then take care that that order is consistent across all membership UAMs used for the PPLD.

3.5.2. REQUIREMENT CHAINS FOR FEASIBLE TOPOLOGIES

The way TFs are generated by the current methods is by selecting components that act as a power/energy source, and others to act as a sink. Then all TPBCs are chosen such that power/energy can go from the sources to the sinks, i.e. the topology is connected [8]. Some more requirements may be added as constraints to reduce the number of TPs to a set that is declared to be the set of TFs. As such, the work done up to section 3.4 matches really closely to what the current state of the art does to generate the topologies.

Here however the approach of using requirement chains is taken to develop the TFs. This method is based on the work done in [20, 27, 21, 22], and was briefly described in section 3.1.3 with the FRDs. With the FRDs however only component types were considered. Here the expanded method is presented, which shows how to develop TFs based on the requirement chain method, while incorporating to it the method to avoid isomorphic topologies.

In summary, components are not chosen apriori to act as sources or sinks. Instead, given an output requirement, the set of components $\mathcal{S}_c$ is explored for components that can deliver the desired output. Given then those components, it is looked at what inputs the components require, and again the set of components $\mathcal{S}_c$ is explored to see what components exist that have an output that can supply those inputs. This is repeated until no more components/instances can be found to satisfy the inputs currently being explored. This is referred to as the requirement chain method. In section 3.5.2.1 a more detailed example of this method is provided, whereas in section 3.5.2.2 it is presented how this method is implemented into TopoGen in combination with the method that avoids developing isomorphic topologies.

3.5.2.1. EXAMPLE FOR REQUIREMENT CHAINS METHOD

In Figure 3.18 and in Figure 3.19 an example of the requirement chains method is shown. In Figure 3.18 four components are shown, each with only one instance. Also the output requirement R_o is shown. The output ports are labelled as $[y_1..y_5]$, while the input ports are labelled as $[u_1..u_5]$. The dotted arrows show the local map of each instance, i.e. on what input ports each output port depends on. In Figure 3.19 both the local map UAM and the compatibility UAM are provided.

To form then a requirement chain, first, all components that deliver an output that matches the output requirement R_o are retrieved. It turns out that y_1 is a port that can deliver the required output, and thus this component is chosen as the first component of the requirement chain. Using then the local map, it is found that for the output port y_1, the inputs u_1 and u_2 need to be fed. With this, the second step in the requirement chain iteration starts: Now one must find components that can deliver an output that matches with u_1, and again for u_2. The components can be found by making use of the compatibility UAM, as shown in Figure 3.19. The input port u_1 can only be matched by the output port y_3. However the input port u_2 can be matched by the output ports y_4 and y_5. This is referred to as a branching in the requirement chain.

The process continues then with the third step in the requirement chain iteration, where now output ports to match the input ports u_3, u_4 and u_5 need to be found. It turns out that for u_3 and u_4 no components exist that can supply these input ports. As such the requirement chain stops there. On the other hand it is found that y_2 from the first component can supply u_5. Thus a feedback loop is established. The local map is used again to see what new input ports there are to explore. However at this stage all input ports shown in Figure 3.18 have already been visited

and connected. Thus the iteration can stop, completing the requirement chain. As is explained further in section 3.5.2.4, in this example the requirement chain represents two TFs.

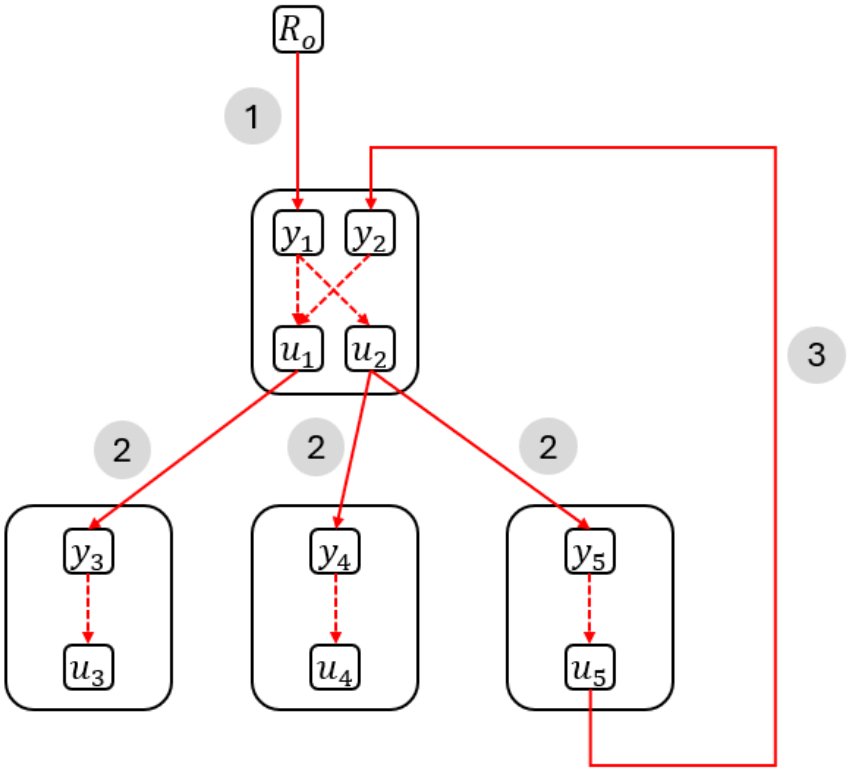


Figure 3.18: Example to illustrate the steps taken to create TFs using requirements chains.

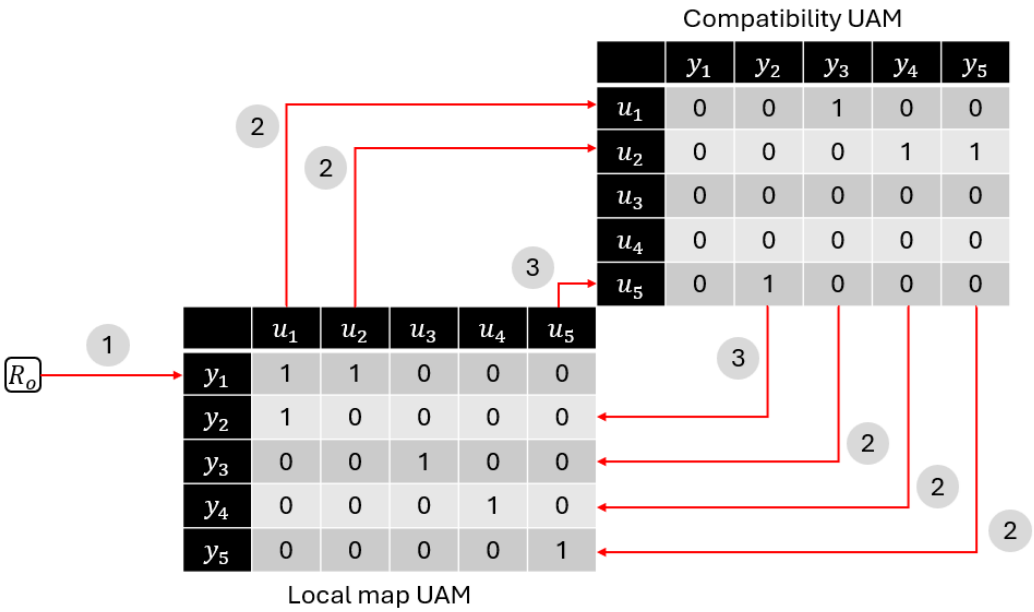


Figure 3.19: How the local map matrix and the compatibility matrix are used to develop TFs using requirement chains.

3.5.2.2. METHOD FOR REQUIREMENT CHAINS WHILE AVOIDING ISOMORPHIC TOPOLOGIES

Having described how requirement chains are formed and how to avoid creating isomorphic topologies, here it is summarised how the two were put together. In TopoGen, the script that runs the combination of these two methods is found under `generate_topologies/calculate_Tf.py`.

Given that a lot of time had gone into putting together the setup that was described in section 3.1 and developing the program TopoGen, little time remained for the research into developing TFs while avoiding isomorphic topologies. Due to the success that was achieved with the CPG from section 3.4.2.2, the requirement chain method was put together to work in a similar fashion. Unlike the work that was presented in section 3.3 and in section 3.4, which maximise the use of matrices, the method here constructs the requirement chains by dealing with each port in `u_ports2explore` one by one. As is shown in sections 3.5.3 to 3.5.5, this approach is sufficient to answer the MRQ. However there is likely room for improvement in how the requirement chains are produced. In section 4.1.2.5, part of the list of recommendations to consider for future work, an alternative method that uses matrices to produce requirement chains is shown. More work and time would be required though before this alternative can be used. Meanwhile, here, the CPG-inspired requirement chain method is presented, with the results obtained from this approach presented in sections 3.5.3 to 3.5.5.

First, the list of unit types [V-ID] is offered to the user. The user selects one of these to specify the output requirement R_o. From here on, the following steps are taken to initialise the requirement chain method:

1. Find ports that can both deliver the selected output requirement and are output ports:

$$\begin{aligned}\mu\left([\text{R-ID}|\text{V-ID}(R_o).\text{N-ID}(y)]\right) &= \mu([\text{R-ID}|\text{V-ID}(R_o)]) \wedge \mu([\text{R-ID}|\text{N-ID}(y)]) \\ &=: \mu([\text{R-ID}(R_o)])\end{aligned}$$

2. Find instances with those compatible ports, i.e. $\mu([\text{H-ID}|\text{R-ID}(R_o)])$.
3. Find components with those compatible ports, i.e. $\mu([\text{C-ID}|\text{R-ID}(R_o)])$.
4. Put together $\mu([\text{C-ID}|\text{R-ID}(R_o)] \times [\text{H-ID}|\text{R-ID}(R_o)])$ to then find first available instance for each component, as was shown in section 3.5.1.
5. Initialise number TFs by the number of components that have been found to deliver the desired output requirement. (More on this in section 3.5.2.3.)
6. Obtain the input ports of the first instances that have been selected and insert this into the set of input ports to explore, i.e. `u_ports2explore`.

The following sets are used in TopoGen in order to create the requirement chains:

- `seen_y_ports`: A set of output ports IDs that specifies which output ports have been visited by the requirement chain.
- `seen_u_ports`: A set of input ports IDs that specifies which input ports have been visited by the requirement chain.
- `set_current_y_ports`: A set of output ports IDs that specifies which output ports have been visited in the current iteration step. These get added at the end of the iteration to the

set `seen_y_ports`. Otherwise, when iterating over each input port and looking for what output ports are available for a matching, some output ports may be left out because they were already considered for a previous input port.

- `u_ports2explore`: A set of input ports IDs, specifying which input ports to explore next.
- `explored_u2ny_connections`: Recording the requirement chain by specifying the connections from input ports to output ports as `(input-port-ID, output-port-ID)` tuples.
- `explored_y2u_connections`: Recording the relevant local map connections that specify which input ports each output port depends on as `(output-port-ID, input-port-ID)` tuples.

Then the iteration shown in the Pseudo-code listing 3.1 is followed to create the requirement chain. On line 19 the condition to choose the next instance is shown. Similarly, on line 25 the stopping criterion for the requirement chain is shown. As it was described in section 3.5.2.1, the requirement chain stops at an instance if all input ports of that instance have already been explored. If not, then as shown in the next lines up to line 30, the input ports that have not yet been explored get added to the set `u_ports2explore`. Once the iteration is completed, the results are produced. In section 3.5.2.3 it is explained how these are extracted and look like.

Listing 3.1: Pseudo-code of how feasible topologies are generated

```
# Initialisation:
- compile list of units that appear in output ports
- request single output requirement from user using list of units from ...
    output ports
- find output ports, components and instances that deliver output ...
    requirement
- choose first instance from components that deliver output requirement
- initialise number topologies by number of components that deliver ...
    output requirement
- obtain u_port and insert into set of u_ports2explore.

# Iteration:
- while u_ports2explore not empty:
    - initialise set_current_y_ports as empty set
    - for each u_port in u_ports2explore:
        - seen_u_ports.update(u_port)
        - u_ports2explore.discard(u_port)
        - explore to which components current u_port leads to
        - for each component:
            - for each instance:
                - obtain next_y_port that current u_port leads to (only ...
                    one per component)
                - if next_y_port in seen_y_ports: # criterion to choose ...
                    next instance
                    - continue to next instance
                - else:
                    - set_current_y_ports.update(next_y_port)
                    - explored_u2ny_connections.update((u_port,y_port)) ...
                        # establish connection to instance with y_port
                    - obtain next_u_ports from current next_y_port
                    - if all(next_u_ports) in seen_u_ports: # stopping ...
                        criterion requirement chain
                        - break, explore next component
                    - else:
                        - add (next_u_ports - seen_u_ports) to ...
                            u_ports2explore
                        - record local map from next_y_port to ...
                            next_u_ports to explored_y2u_connections
                        - break to explore next component
    - seen_y_ports.update(set_current_y_ports) # update what ports have ...
        been visited in this step of the iteration before moving to the ...
        next one.

# Results:
- calculate number topologies from explored_connections
- present results
```

3.5.2.3. PRESENTING THE RESULT OF THE REQUIREMENT CHAIN

Once the iteration is complete, the results are contained within the two sets `explored_u2ny_connections` and `explored_y2u_connections`. Of these two especially the set `explored_u2ny_connections` matters, since it presents the connections made from one instance to another. The set `explored_y2u_connections` presents the relevant local maps, i.e. the connections within some instance, which is presented for the sake of completion.

In the program TopoGen the two sets are converted into UAMs with more readable labels. In Table 3.8 the UAM of the set `explored_u2ny_connections`, here defined as Q, is presented. In the last square brackets are the IDs of the ports that are being connected. The first square brackets show the instance number. Here all components only had one instance, thus all square brackets show that only the first instance has been selected, i.e. '[0]'. Finally, the alias of the component to which each port belongs to is shown.

As can be seen when comparing Table 3.8 with the compatibility matrix shown in Figure 3.19, Q consists of rows and columns that have been extracted from the compatibility matrix. Depending on the set of components $\mathcal{S}_c$ and the set of requirements $\mathcal{S}_r$, different requirement chains are produced, and thus also different rows and columns would be extracted from the corresponding compatibility matrix.

	Component_y1y2u1u2 [0] [y_2]	Component_y3u3 [0] [y_3]	Component_y4u4 [0] [y_4]	Component_y5u5 [0] [y_5]
Component_y1y2u1u2 [0] [u_1]	0	1	0	0
Component_y1y2u1u2 [0] [u_2]	0	0	1	1
Component_y5u5 [0] [u_5]	1	0	0	0

Table 3.8: UAM Q for the example shown in Figure 3.18 and Figure 3.19.

3.5.2.4. COUNTING NUMBER OF FEASIBLE TOPOLOGIES

A last challenge remains, which is to calculate the number of TFs from the requirement chain that has been produced. Two terms need to be calculated:

- **Number of components that meet output requirement R_o:** This is done by computing the length of the vector $\mu_r([\text{C-ID}|\text{R-ID}(R_o)])$. This is shown in (3.27), where the operator $\ell(\cdot)$ specifies the length of the term in its brackets, and the subscript 0 specifies that the rows need to be counted. (A subscript 1 would specify that columns are to be counted.)

$$|\mathcal{T}_{f,r}| := \ell_0\left(\mu_r([\text{C-ID}|\text{R-ID}(R_o)])\right) \tag{3.27}$$

  In the example shown in Figure 3.18 and Figure 3.19, there was only one component that would deliver the required output. Therefore, in that case, $|\mathcal{T}_{f,r}| = 1$.

- **Number of branches in requirement chain:** Some rows in Q will have more than one '1' in it. This indicates the presence of a branching in the requirement chain. Each branch represents a different TF. In the example shown in Figure 3.18 and Figure 3.19, a branching of two branches is found. When looking at the Q that was obtained from the example, shown in Table 3.8, indeed there exists a row with more than one '1' in it. To count the number of TFs that result because of the branching of the requirement chain, for each branching that appears in the requirement chain, i.e. for each row with more than one '1' in it, the sum is computed. Each sum is corrected by -1, since a single branch would have represented no branching. Therefore, the sum of all branches can be calculated using (3.28).

$$|\mathcal{T}_{f,q}| := \sum_{i=1}^{\ell_0(Q)} \left(\sum_{j=1}^{\ell_1(Q)} q_{i,j} \right) - 1 \tag{3.28}$$

Having computed $|\mathcal{T}_{f,r}|$ and $|\mathcal{T}_{f,q}|$, the number of TFs can be calculated as shown in (3.29).

$$|\mathcal{T}_f| = |\mathcal{T}_{f,r}| + |\mathcal{T}_{f,q}| \tag{3.29}$$

3.5.3. FEASIBLE TOPOLOGIES FOR THE 3BT-3EM TEST

In the following section the results for the 3BT-3EM test are presented. (See section 3.1.3 for a description of all tests.)

To start off, Table 3.9 presents the TFs that were found when requesting torque as the output requirement R_o. From the set of components $\mathcal{S}_c$ for the 3BT-3EM test, the three electric motor components can deliver such requirement. Thus $|\mathcal{T}_{f,r}| = 3$. Then, as shown in Table 3.9, each first instance of the electric motor components connects to the first instances of each battery component in the set of components $\mathcal{S}_c$. Therefore $|\mathcal{T}_{f,q}| = 2+2+2 = 6$, and $|\mathcal{T}_f| = 3+6 = 9$.

	Battery_LeadAcid_180Ah [0] [01HYG032PAXZA2SQM598C94VKR]	Battery_LeadAcid_320Ah [0] [01HYG033VKN20C8GMC4FRN2AHG]	Battery_LeadAcid_240Ah [0] [01HYG0337X4J3YXTAK4WPGC8C4]
EMotor_HPEVS-AC-9 [0] [01HYG036JJ7PF8R0W0PWY72EE9]	1	1	1
EMotor_HPEVS-AC-12 [0] [01HYG034GHH45KC4GYJP3SN5YZ]	1	1	1
EMotor_HPEVS-AC-20 [0] [01HYG035FY42SAHKK0033R8MMH]	1	1	1

Table 3.9: TFs for the 3BT-3EM test.

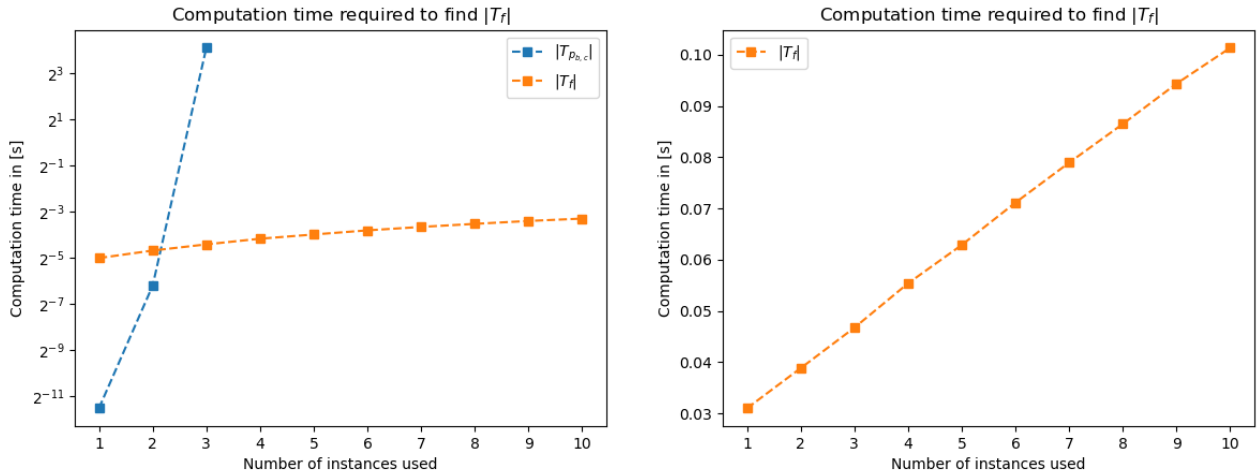
In Table 3.10 the number of TFs and the computation time to compute the TFs is presented, alongside the results obtained from the previous sections. When comparing the number of TPBCs $|\mathcal{T}_{p,bc}|$ to the number of TFs $|\mathcal{T}_f|$, only a minuscule $0.000'7\,\%$ of the TPBCs are found to also be TFs. This is evidence of how massively constrained the problem of topology generation actually is. TFs are the rare exception to all the TPs that exist.

	Base-2	**Base-10**	**Computation time [s]**
$\lvert\mathcal{T}_p\rvert$	2^{576}	2.47×10^{173}	-
$\lvert\mathcal{T}_{p,b}\rvert$	2^{65}	4.61×10^{19}	-
$\lvert\mathcal{T}_{p,bc}\rvert$	2^{14}	13.3×10^{3}	0.012
$\lvert\mathcal{T}_f\rvert$	$2^{3.2}$	9	0.038

Table 3.10: Final results for 3.2BT-3.2EM test.

In Table 3.10 also the computation time required to find all TFs is shown. Comparing the computation time with the times reported in the literature (see section 2.2.2), it is a massive achievement that the TFs are computed well under a second. Still, when only looking at the results of Table 3.10, it seems like the computation time required to compute all TFs is greater than to find all TPBCs. However Table 3.10 only shows the results for the 3.2BT-3.2EM test. When performing the entire 3BT-3EM test, then the results shown in Figure 3.20 are obtained.

In Figure 3.20a the computation time required to compute both the TPBCs and the TFs is shown with respect to the number of instances used. When using only one or two instances for the 3BT-3EM test, the CPG method used to compute the TPBCs is significantly faster. However, without being able to reject the isomorphic topologies and disregarding the set of requirements $\mathcal{S}_r$, the computation time quickly increases with an increasing number of instances. At three instances per component, the computation time for the TPBCs is already at 17 s. Additionally, the growth for the TPBCs seems to be quadratic on the log-2 scale, as was also shown in section 3.4.2.2. That



(a) Computation time required to compute TPBCs (in blue) and TFs (in yellow) versus the number of instances used in the 3BT-3EM test.

(b) Computation time required to compute TFs (in yellow) on a linear y-scale versus the number of instances used in the 3BT-3EM test.

Figure 3.20: Computation time required to compute all TFs.

means that computing the TPBCs with four instances per components will already require having to wait minutes if not hours. Considering any more instances is no longer practical. This matches closely with the results that were reported in the literature.

A very different story is told when looking at the computation time required for the TFs. As shown in Figure 3.20a, unlike the quadratic growth in computation time for the TPBCs, the growth for the TFs is significantly lower. Figure 3.20b plots the computation time no longer on a log-2 scale, but on a linear scale, and shows that the computation time grows linearly with the number of instances considered. Even when considering 10 instances for the 3BT-3EM test, which amounts to a total of 60 instances, all nine TFs are found within a tenth of a second.

The results obtained here indicate that the 3BT-3EM test has been passed. This is because despite increasing the number of instances per component, the number of TFs remains constant at nine, always producing the results shown in Table 3.9. Additionally, because of the method that avoids isomorphic topologies, the impact on computation time is minimal, increasing linearly as the number of instances is increased.

In the next section the results for the ISO-car test are shown, which tests more rigorously the method used to avoid isomorphic topologies.

3.5.4. FEASIBLE TOPOLOGIES FOR THE ISO-CAR TEST

In the following section the results for the ISO-car test are presented. (See section 3.1.3 for a description of all tests.)

First the ISO-car test was performed on a set of components that has one instance for both the car and the electric motor, but two instances for the batteries. This is referred to as the ISO-car 1.2BT test, and only one TF is expected when requiring thrust as an output requirement R_o. Indeed, only one TF is found, as shown in Table 3.11, obtained within 2.13×10^{-2} s. This shows that the method used to develop TFs does consider using additional instances when necessary and when these are available.

The ISO-car test is repeated then to consider now two batteries, each with two instances, i.e. the ISO-car 2.2BT test. Here four TFs were expected. These are listed below, where the first battery is the battery that is directly connected to the car, and the second battery is the one that connects to the electric motor:

1. Battery_LeadAcid_180Ah [0]; Battery_LeadAcid_180Ah [1]
2. Battery_LeadAcid_180Ah [0]; Battery_Lithium_14kWh [0]
3. Battery_Lithium_14kWh [0]; Battery_LeadAcid_180Ah [0]
4. Battery_Lithium_14kWh [0]; Battery_Lithium_14kWh [1]

Instead of obtaining these four TFs though, only three are obtained. These are computed within 2.80×10^{-2} s, and the results are shown in Table 3.12. The TFs that are obtained now are:

1. Battery_LeadAcid_180Ah [0]; Battery_LeadAcid_180Ah [1]
2. Battery_LeadAcid_180Ah [0]; Battery_Lithium_14kWh [1]
3. Battery_Lithium_14kWh [0]; Battery_Lithium_14kWh [1]

While the TFs that are produced are indeed feasible and are not isomorphic, there are two mistakes to be found:

- The TF where first the lithium battery is connected and then the lead-acid battery is missing.
- The second TF uses the second instance for the lithium battery even though it could have used the first one.

The reason these mistakes appear is because of the method that was used to produce the requirement chains. This involved developing the requirement chain in the `explored_u2ny_connections` set. Using this method the requirement chain will develop a first branch to connect either the lead-acid or the lithium battery to the car, and then proceed developing the rest of the requirement chain. However it will consider to already have connected to the lithium battery, even if the topology used the lead-acid battery for the car. This is why for example the second TF that was obtained connects not the first but the second instance of the lithium battery to the electric motor. The problem therefore lies in that currently, by only working with one set, i.e. the `explored_u2ny_connections` set, there is no way to check which components/instances have already been used for one topology.

This can be adjusted. Instead of working with only the set `explored_u2ny_connections`, each time a branching is found, the set should be copied to as many branches as are found in the current iteration step. Then each set is developed further to complete the requirement chain. Each set will then represent one TF, and it should be possible to ensure that then all four TFs mentioned above are obtained when running the ISO-car 2.2BT test. Due to lack of time, this could not be developed and tested, and thus needs to be considered in the future.

Therefore, as it stands, the ISO-car test is not passed. However some of the results it produces are correct, such as for the ISO-car 1.2BT test, and are already enough to give an insight into the impact avoiding isomorphic topologies has on the computation time, as requested by the MRQ.

	Battery_LeadAcid_180Ah [1] [01HZW6QCHXRHJWBSN87XTHXMD3]	EMotor_HPEVS-AC-12 [0] [01HZW6QCWCJCW4P7RG5M7T8D8W]	Battery_LeadAcid_180Ah [0] [01HZW6QCGBYF93B9A79GTCRT67]
Car_ISO [0] [01HZW6QCPD0WTS0XXS5N2F0X79]	0	0	1
EMotor_HPEVS-AC-12 [0] [01HZW6QCWCA7JADN078PGXZ0ZY]	1	0	0
Car_ISO [0] [01HZW6QCPDSJ0S9HCFQ3H92C7B]	0	1	0

Table 3.11: TFs for the ISO-car 1.2BT test, computed in 2.13×10^{-2} s.

	Battery_Lithium_14kWh [0] [01J08EYK8RK9WS8ADE0NS53HFX]	Battery_Lithium_14kWh [1] [01J08EYKA42V8V8CG401ACNSP3]	EMotor_HPEVS-AC-12 [0] [01J08EYKKKRNAZ2J1AB17D72X8]	Battery_LeadAcid_180Ah [1] [01J08EYK5R37RDE6Y6B0V17MX0]	Battery_LeadAcid_180Ah [0] [01J08EYK3PW6S27V8R4KPV222R]
EMotor_HPEVS-AC-12 [0] [01J08EYKKK43KDZEX8VHRMWN0K]	0	1	0	1	0
Car_ISO [0] [01J08EYKDVXJ4N4D8PMMTJ6090]	1	0	0	0	1
Car_ISO [0] [01J08EYKDVMR63R83EBPTDC7ZK]	0	0	1	0	0

Table 3.12: TFs for the ISO-car 2.2BT test, computed in 2.80×10^{-2} s.

3.5.5. FEASIBLE TOPOLOGIES FOR THE LOOPS & BRANCHES TEST

In the following section the results for the Loops & Branches test are presented. (See section 3.1.3 for a description of all tests.) Unlike the tables from the previous tests, which show the requirement chains that were obtained, here the tables have the ULIDs for the ports removed in order to make the results more readable. The results that TopoGen spits out will however always include these. This is also the reason why some components appear multiple times: The same instance but a different port is being considered.

In a first test only a single instance is used for all components for the truck. It is expected that when thrust is requested as an output requirement R_o, two TFs are produced. One of the TFs is the tractor powered directly by an electric motor, whereas in the second case a gearbox is placed in between. Indeed, as is shown in the results in Table 3.13, these two TFs are produced. The computation time required to put together these two TFs was 3.19×10^{-2} s. In Table 3.13 it can be observed that even the driver, i.e. the controller, is connected appropriately.

Next the Loops & Branches test is repeated in which the set of components contains two different gearbox components, but still all components have one instance. The expectation now is to obtain three TFs, where one TF makes use of no gearbox, one makes use of one gearbox, and the third one makes use of the second gearbox. Again, as can be verified when looking at Table 3.14, these are indeed also the results that are obtained, all within 3.93×10^{-2} s.

However when the previous test is repeated, but with two instances for each component, then suddenly five (non-isomorphic) TFs are produced. The results are shown in Table 3.15, and were obtained in 9.88×10^{-2} s. As can be read from Table 3.15, the same three TFs from the previous test were obtained. However two more TFs appear, which connect the gearboxes one after the other before connecting to the electric motor. Also the second instance of some components is used, when this is not necessary. Something similar happens when only one gearbox is considered, all else same. (In that case six TFs are produced.) The hypothesis is that these errors appear, because the gearboxes are an unusual component that have an input port that is identical to its output port. Even with a constraint that specifies that a component cannot connect to another component of the same component type, the method used here to produce the requirement chain seems to bypass this constraint.

To verify this hypothesis, a final test has been performed, in which now no gearbox component is used. Instead two electric motor components are used, and all components have two instances. The expectation is that only two TFs are produced, where each TF uses one of the two electric motors. Indeed, as shown in Table 3.16, these two TFs are obtained. The results were computed in 4.35×10^{-2} s.

The last test confirms the hypothesis that the gearbox component is responsible for causing the mistakes. As such the Loop & Branches test can only be passed when components such as gearboxes are excluded. More work is required to improve how the TFs are generated in order to be able to handle a larger variety of components.

Driver_MPC [0]	Trailer_GoupilG4 [0]	0	0	0	0	0	1
Trailer_GoupilG4 [0]	Driver_MPC [0]	1	0	0	0	0	0
Gearbox_2Speed_(10,15) [0]	EMotor_HPEVS-AC-12 [0]	0	1	0	0	1	0
EMotor_HPEVS-AC-12 [0]	Driver_MPC [0]	0	0	0	0	0	0
Driver_MPC [0]	Trailer_GoupilG4 [0]	0	0	0	1	0	0
Battery_LeadAcid_180Ah [0]	Battery_LeadAcid_180Ah [0]	0	0	1	0	0	0
Circuit_Testing [0]							

Table 3.13: TFs when specifying thrust as an output requirement R_o and using one instance per component for all truck parts for the Loops & Branches test. Results were computed in 3.19×10^{-2} s.

	Circuit_Testing [0]	Driver_MPC [0]	Driver_MPC [0]	Battery_LeadAcid_180Ah [0]	Gearbox_2Speed_(10,20) [0]	Trailer_GoupilG4 [0]	Gearbox_2Speed_(10,15) [0]	EMotor_HPEVS-AC-12 [0]
Driver_MPC [0]	1	0	0	0	0	0	0	0
Trailer_GoupilG4 [0]	0	1	0	0	0	0	0	0
Trailer_GoupilG4 [0]	0	0	0	0	1	0	1	1
Driver_MPC [0]	0	0	0	0	0	1	0	0
Battery_LeadAcid_180Ah [0]	0	0	1	0	0	0	0	0
EMotor_HPEVS-AC-12 [0]	0	0	0	1	0	0	0	0

Table 3.14: TFs when specifying thrust as an output requirement R_o and using two gearbox components, but still only one instance per components for all truck parts for the Loops & Branches test. Results were computed in 3.93×10^{-2} s.

	Battery_LeadAcid_180Ah [0]	Gearbox_2Speed_(10,15) [0]	Gearbox_2Speed_(10,20) [0]	Driver_MPC [0]	Trailer_GoupilG4 [1]	Driver_MPC [1]	Battery_LeadAcid_180Ah [1]	Trailer_GoupilG4 [0]	Gearbox_2Speed_(10,20) [1]	Driver_MPC [1]	EMotor_HPEVS-AC-12 [0]	Circuit_Testing [0]	Driver_MPC [0]	Circuit_Testing [1]	EMotor_HPEVS-AC-12 [1]	Gearbox_2Speed_(10,15) [1]
EMotor_HPEVS-AC-12 [1]	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
Trailer_GoupilG4 [1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Battery_LeadAcid_180Ah [1]	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
Driver_MPC [0]	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
Trailer_GoupilG4 [0]	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
EMotor_HPEVS-AC-12 [0]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Trailer_GoupilG4 [0]	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
Driver_MPC [1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Trailer_GoupilG4 [1]	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
Battery_LeadAcid_180Ah [0]	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Driver_MPC [1]	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Driver_MPC [0]	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
Gearbox_2Speed_(10,15) [0]	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
Gearbox_2Speed_(10,20) [0]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 3.15: TFs when specifying thrust as an output requirement R_o and using two gearbox components, and using two instances per component for all truck parts for the Loops & Branches test. Results were computed in 9.88×10^{-2} s.

	Driver_MPC [0]	Trailer_GoupilG4 [0]	Driver_MPC [0]	Trailer_GoupilG4 [0]	EMotor_HPEVS-AC-12 [0]	EMotor_HPEVS-AC-20 [0]	Battery_LeadAcid_180Ah [0]
Driver_MPC [0]	0	1	0	0	0	0	0
Driver_MPC [0]	0	0	0	1	0	0	0
Trailer_GoupilG4 [0]	0	0	1	0	0	0	0
EMotor_HPEVS-AC-12 [0]	0	0	0	0	1	0	0
EMotor_HPEVS-AC-20 [0]	0	0	0	0	0	1	0
Battery_LeadAcid_180Ah [0]	1	0	0	0	0	0	0
Circuit_Testing [0]					1	0	0
EMotor_HPEVS-AC-12 [0]					0	0	1

Table 3.16: TFs when specifying thrust as an output requirement R_o and using no gearbox components, two electric motor components, and two instances per component for all truck parts for the Loops & Branches test. Results were computed in 4.35×10^{-2} s.

3.5.6. CONCLUSION

In this section the development of TFs from a set of components $\mathcal{S}_c$ and a set of requirements $\mathcal{S}_r = \{R_o\}$ was considered. In section 3.5.1 the method that was used to avoid isomorphic topologies was presented. In section 3.5.2 it was described how to develop the TFs from requirement chains. The strategy put together here was then tested for the 3BT-3EM, ISO-car and Loops & Branches test.

The results show that the method to avoid isomorphic topologies seems effective. None of the results show any isomorphic topologies. Additionally, for simple cases, all tests can be passed. Crucially, all of the TFs are developed in ground-breaking times when compared to the computation times mentioned in literature. In section 3.5.3 it was even shown that as the number of instances is increased, unlike what is typically mentioned in the literature, the computation time only grows linearly. This is only thanks to the method that was applied to avoid isomorphic topologies.

However issues remain. When adding more components and instances, wrong results are produced in the ISO-car test. Also for the Loops & Branches mistakes have been found when the gearbox component was included in the set of components. The issues are caused by the method used to develop the requirement chains. Some ideas already exist on how to improve this method. In section 3.5.4 it is presented how the issues found in the ISO-car test can be avoided in the future. The requirement chain method should also be reviewed to see whether it can be improved upon using matrix calculations instead. Adjustments like these may lead to avoiding the mistakes that are currently appearing in the Loops & Branches test as well.

In conclusion, significant results have been obtained, with which the MRQ can be answered. Still, mistakes in the method presented here have been found, which need to be addressed in future work.

4

CONCLUSION & RECOMMENDATIONS

In chapter 1 Co-Design was introduced, explaining that this approach could develop optimal designs, unlike the more commonly used sequential design process. This would make it therefore the approach to take in order to develop the solutions needed to decarbonise the European Union (EU) transportation sector, and thus meet the EU goal to be carbon neutral by 2050. Unfortunately, despite the strengths of Co-Design, it was also explained that it is currently restricted to working with only a small set of 10-20 components, making it impractical to actually use it as is needed. Focusing on where this limitation comes from, in chapter 2 the current methods around Co-Design were reviewed, revealing that the sub-process of topology generation the reason was as to why Co-Design was restricted to such small sets. This led to defining the main research question (MRQ) and the two supplementary research questions (SRQs). These were presented in section 2.3, and focused on expanding the knowledge in this area of Co-Design, as well as testing whether preventing isomorphic topologies useful is to remove the 10-20 components limitation.

The sections that followed presented the work that has been done towards answering these research questions. In section 3.1 the basic setup for the research was described. In section 3.2 the potential topologies (TPs) when no constraints are applied were considered. In section 3.3 the port-to-port constraint is put in place, while in section 3.4 also the compatibility constraint is added. Finally, in section 3.5, the feasible topologies were considered, while avoiding the emergence of isomorphic topologies.

Here now, in the conclusion of this report, the MRQ and SRQs are put back into focus in order to highlight the contributions that have been made towards these research questions. These are presented as separate sections, each addressing a research question, where the research questions are dealt with in reverse order in order to match the increasing amount of work that was required to answer them. After the conclusion, section 4.1 is provided, which wraps up the report with a list of recommendations for future work on the topic of topology generation for Co-Design.

4.0.1. CONTRIBUTIONS TOWARDS SRQ-2

In SRQ-2, it was asked how much each of the constraints that were used here upper bound the number of feasible topologies (TFs) on a given set of components $\mathcal{S}_c$. With the work done here, it was possible to answer the SRQ-2, by specifying how much each constraint reduces the number

of TPs.

In section 3.3 it was first explained why a port-to-port constraint should be considered, and then, by applying it, it was shown that the port-to-port constraint eliminated the majority of the potential topologies (TPs) already. Namely, for the 3.2BT-3.2EM test, while originally there were 2.47×10^{173} TPs, with the port-to-port constraint applied, 4.61×10^{19} potential topologies with bijective connections (TPBs) remained. This work led to developing the expression shown in (3.11), a significantly better upper bound for any set of components $\mathcal{S}_c$ compared to the typical upper bound shown in (3.8) and usually mentioned in literature.

In section 3.4 a further constraint was developed, which considered the compatibility between the ports. With the port-to-port and the compatibility constraint applied, for the 3.2BT-3.2EM test, the TPBs were reduced further to 13.3×10^3 potential topologies with bijective compatible connections (TPBCs). When exploring the number of TPBCs with r connections, similar results as to the ones reported in the literature were obtained. To produce the TPBCs, first SAT solvers were used as opposed to the typically used CSP solvers. This was hinted to be done in [7], but in section 3.4 it was explained further why this is indeed the right approach. However finding that despite expectations SAT solvers were slow to produce the TPBCs with the constraints that were applied so far, an alternative to develop the TPBCs was explored. It was discovered that a constrained permutation generator (CPG) could significantly outperform the SAT solvers, at least with the constraints that were taken into account. Using square UAMs, it was shown that for a UAM of size two, the CPG was about 128 times faster than the SAT solver. Increasing this to a UAM of size six, and then the CPG would compute the TPBCs 4′096 times faster. The computation time would therefore increase at a shallower rate when using the CPG as opposed to the SAT solver. This is an important discovery, because in literature so far only CSP/SAT solvers were considered. In section 3.4 it was hypothesised as to why the CPG could outperform the SAT solver so much, but further research is required in order to exactly explain how this performance difference comes to be, as is mentioned in the list of recommendations in section 4.1.2.2.

Finally, in section 3.5, the TFs were put together. Here a method was implemented that avoided the emergence of isomorphic topologies. Thanks to this approach, the expected 9 TFs from the 3.2BT-3.2EM test were obtained. While already mentioned in the literature, this provides further evidence of how TFs are the exception to all the TPs that exist, i.e. how enormously constrained the problem of topology generation actually is.

4.0.2. Contributions towards SRQ-1

In SRQ-1 it was asked how much the use of (unweighted) adjacency matrices (UAMs) would reduce the number of constraints needed to solve the topology generation problem.

In literature, as was explained in chapter 2, UAMs are barely made use of, even though nodes and edges are. The work done here therefore stands out from the literature, by approaching the topology generation problem with very much the use of UAMs. By expanding the port-level-description (PLD) from [7] to the property-port-level-description (PPLD), as well as structuring the topology generation problem with the interconnected-system-model (ISM) and making use of boolean compositions to manipulate the data stored in the UAMs, a successful setup based on UAMs was put together with which the topology generation problem could be studied.

This setup with the UAMs allowed to find important UAM properties, which were described in section 3.3. The use of UAMs also made it possible to recognise how to use combinations and

permutations to develop (3.11) to find the number of TPBs. When developing both the port-to-port and the compatibility constraint, UAMs made it possible to express these in straightforward UAM expressions, instead of the many single constraints that have to be expressed with the methods described in the literature. In section 3.3, where the TPBCs were developed, thanks to the use of UAMs, the alternative to SAT solvers, the CPG, was discovered, which significantly outperformed SAT solvers with the constraints that were considered. Even when developing the strategy to avoid isomorphic topologies or the requirement chain method needed to put together the TFs, as described in section 3.5, the UAMs continued to provide valuable insight to assemble these methods.

Most of all, and answering the SRQ-1, the use of UAMs has gotten rid of the need to have to rewrite any of the constraints when dealing with a new set of components $\mathcal{S}_c$ or a new output requirement R_o. This way, the risk of forgetting any constraint has been eliminated. The setup with UAMs also made it possible to automate the entire process of assembling some desired set of components, to applying the constraints, to finally generating the TFs upon specifying some output required R_o. This was all implemented into the TopoGen program, which was developed as part of the research work and is described in section 3.1.4. The result is that unlike the literature that has been reviewed, with the UAMs the program TopoGen is capable of producing TFs for a wide range of different sets of components $\mathcal{S}_c$ and output requirements R_o. The work presented here therefore outlines how indispensable UAMs are when dealing with topology generation for Co-Design.

4.0.3. Contributions towards MRQ

The main goal of the work done here was to tackle the MRQ, which asked how extending the description of components while avoiding isomorphic topologies would influence the computation time to generate all TFs in comparison to the current methods applied in the literature.

This required to first develop a setup with which components could be described and sets of components could be put together. This basic setup was dealt with in section 3.1 and was implemented into the TopoGen program. Section 3.1.4 provides details on this program and the system on which it was run.

Once the basic setup was put together, and the necessary operations could be performed, three different tests were developed. These were put together for the purpose of evaluating the results obtained from TopoGen when asked to generate the TFs (without producing isomorphic topologies) for a set of components $\mathcal{S}_c$ and a single output requirement R_o. The 3BT-3EM test would test if TFs without isomorphic topologies could be generated when multiple instances per component were available, but only the first instances would be necessary. The ISO-car test on the other hand considers the situation where the extra instances are necessary to form the TFs. Finally, the Loop & Branches test would expose to TopoGen program to sets of components, where the TFs would have feedback loops and branching. (See more on these tests in section 3.1.3.) As such the work presented here stands out by that from the literature not just because TFs for multiple sets of components and output requirements could be generated, as discussed in section 4.0.2, but also by the tests that have been put together to evaluate the results. Still, these tests may not be enough to guarantee the approach used here to generate the TFs can work with any other set of components or set of components. Further research is needed to see how such evidence may be provided, as is highlighted again in the list of recommendations, in section 4.1.2.1.

Moving on to the generation of TFs, first, to avoid producing isomorphic topologies, the candidate method described in section 3.5.1 was put together. Next, to produce TFs, the requirement-chain method was developed, inspired by existing literature and talked about in section 3.5.2. The two methods were then combined in order to generate TFs without producing isomorphic topologies.

The results show that for the simplest versions of all three tests, all TFs without any of the isomorphic topologies are generated. Crucially, with the method used here, significantly smaller computation times are needed than the ones mentioned in the literature. Moreover, when performing the 3BT-3EM test for different numbers of instances as is shown in Figure 3.20, computing all TPBCs using the CPG, the method that matches closest with what is found in literature, the computation time grows more or less at the worrying rate of 2^{H^2} as the number of instances H is increased. However when computing TFs while avoiding the isomorphic topologies, the computation time now grows linearly. Already only at three instances the CPG requires 17 s, while the TFs are generated in just 0.05 s. Considering more instances would have required to wait minutes if not hours when using the CPG to develop the TPBCs, but with the method used to generate the TFs, it was no problem to consider many more instances. When dealing with 10 instances per component, the TFs are generated in just 0.10 s.

The work presented here therefore shows that avoiding the isomorphic topologies significantly reduces the computation time required to generate all TFs, so much, that the restriction to sets of components of only 10-20 components no longer seems to apply. This despite using a description for the components that uses more detail than what has been used so far in the literature.

Still, as presented in section 3.5.4 and section 3.5.5, the method implemented here to generate the TFs without isomorphic topologies has its limitations. When more involved cases were considered for the ISO-car test and the Loops & Branches test, mistakes were found. Either not all TFs were computed, or some of the TFs would connect to the wrong instances.

However, as discussed in those sections, the origins of these mistakes have been identified. Additionally, in the list of recommendations presented hereafter in section 4.1, it is also explained how these mistakes may be eliminated in the future. It therefore seems promising that with further work and research, a Co-Design method can be put together that is not limited by the 10-20 components, but can consider significantly higher number of components. Once this would be achieved, Co-Design could be applied to develop solutions needed to decarbonise the EU transportation sector and meet the goal of carbon neutrality by 2050.

4.1. RECOMMENDATIONS FOR FUTURE WORK

In the following a list of recommendations is presented, which is based on the work that was done here, and may lead to additional useful contributions towards topology generation for Co-Design.

In section 4.1.1, recommendations towards the PPLD are given. This is followed then by section 4.1.2, where ideas on how to improve the generation of feasible topologies are provided.

4.1.1. IMPROVEMENTS TO THE PPLD

4.1.1.1. BUILDING SET OF COMPONENTS

In the TopoGen program, instead of first creating the properties, and then assigning them the types, the process actually works the other way around. The components are described by their types, and from this description the appropriate number of properties are created and allocated. However this means for example that there cannot be two output properties with the same unit, as otherwise, in the current version of TopoGen, the two output properties would both get the same property ULID. This limitation can be avoided by developing a user interface, where properties can be added to components. Each time a property is created, the types of that property can be specified.

4.1.1.2. MORE CONSISTENCY FOR THE PPLD

In the current PPLD, the node, unit and domain types are used. However with the work that was done here, it does not seem necessary to have both the unit and domain types. Rather it seems more straightforward to only have one type. For example, instead of having a property with a unit type 'voltage' and a domain type 'electric', it should be sufficient to have one 'voltage - electric' type. This however also raises the question of what exact data should be used to differentiate between properties in such a way that a large variety of components can be described.

Additionally, currently, the node, unit and domain types are entered manually in the description of each component. A lot of attention was required to make sure that the naming of the same type was consistent across components. Otherwise unit types like 'voltage' and 'Voltage [V]' may appear as two different unit types, when in reality they are the same one unit type. A method therefore needs to be developed to ensure consistency across all types.

Finally, there still exists some ambiguity as to what exactly is a component type. These are also not really defined in the literature [22, 20, 21, 27]. Here it was decided that components are differentiated by the unit types the components have. (See section 3.1.2.3.) However other stricter or less strict definitions could be developed. Which one is the most appropriate would need to be explored. It may also be interesting to instead not group components into component types, but rather simply define how similar two components are. This may help in getting rid of the ambiguity in the definition of component types. It may also help to retrieve relevant components faster, where the similarity between components can be exploited. The similarity between components is currently not exploited when developing the requirement chains.

4.1.1.3. EXPAND THE COMPONENT DESCRIPTION

As was described at the beginning of chapter 3, one of the key philosophies in the work that was done here was to maximise the use of the data that is available for each component. The work done here makes a significant step towards that direction with the PPLD in comparison to what has been done so far. However only the surface has been scratched of all that can be done with

the data that is available about each component. Below two examples of further data that can be exploited is mentioned:

- **Saturation limits:** Each component provides certain outputs and requires to be fed certain inputs. However more than that, there exist limits on how much the component can provide or how much of the inputs it can be fed. For example an electric motor can only handle a maximum voltage as an input, and can similarly only provide a maximum output rpm.
- **Stability:** When connecting different components together, it is important to check that such an interconnected system operates as desired. Models can be built for each component, and using the ISM described in section 3.1.1, different topologies can be tested.

Expanding the component description should lead to more constraints, with the result that fewer TFs should exist, but also providing for all TFs significantly more detail.

4.1.1.4. MORE FLEXIBILITY FOR CONNECTIONS

Currently, each port can only connect to one other compatible port. However, in practice, it is possible that two separate ports can also be connected as if they were a single port. This flexibility currently does not yet exist, but may provide TFs that are otherwise ignored.

4.1.1.5. MINIMISING DATA STORAGE OF THE SPARSE UAMS

Currently, all the data that is produced using the PPLD to describe components is stored as UAMs. However, as was described in section 3.3.2, all UAMs used here are either injective, surjective or even bijective. This means that the UAMs are sparse matrices, with the majority of the values stored in the UAM being zeroes. Not just that, but the number of values stored in for example a membership UAM μ([P-ID] × [P-ID]) would increase quadratically. (See proof below.) Indeed, it was noticeable how for the larger sets of components, more time was required by TopoGen to build the sets of components. While it has not become a problem yet with the sets of components that were considered in this work, since even the largest sets of components were built in a few seconds, for even larger sets of components this could quickly require having to wait hours or days.

Proof. Consider a UAM $M \in \mathbb{B}^{p\times q}$. Then there are pq values stored in M. If $p = q$, i.e. M is a square UAM, then there are p^2 values. Therefore, if p values are required for the [P-ID] list, then p^2 values are required for μ([P-ID] × [P-ID]) UAM. ■

If instead the (column) indices of where the 1's are located in the UAM, as it was done with the UALs in section 3.4.2.2, then because all UAMs are either bijective, surjective or injective, they can be stored as a list of numbers. (See proof below.)

Proof. If a UAM is bijective, then there is only one 1 in every row. Thus taking the column indices one ends up with a list of column indices. (And not a list of lists.)

The same is obtained with injective UAMs.

With surjective UAMs, the transpose can be taken to obtain an injective UAM. Then this UAM can also be saved as a list of column indices. An extra value can be added to the UALs to indicate that the transpose has been taken. ■

Unlike with UAMs, which grow quadratically, UALs grow linearly. (See proof below.) Thus, because UALs require less data, large sets of components are built much faster using these.

Proof. Consider a UAM $M \in \mathbb{B}^{p \times q}$. Because M is bijective, there is only one 1 in every row (and column). Thus when storing the column indices, i.e. a UAL, one ends up with a list of length p. Unlike with a UAM where pq values are needed, with a UAL only p values are needed.

Again a value can be added to the list, which indicates whether the transpose has been taken or not. That way one can choose for bijective UAMs to store either a UAL of length p or of length q. ■

However converting all UAMs to UALs will also require adjusting the operations that were done with the UAMs. Operations like boolean composition or negation of the UAM will have to be translated to work with UAL instead. How to achieve this needs to be explored, but also what impact this has on the computation times to complete these operations. After all, the boolean composition that was applied to UAMs equates to an efficient matrix multiplication, as was shown in section 3.1.2.4.

4.1.2. IMPROVEMENTS FOR THE GENERATION OF TFS

4.1.2.1. REVIEW VALIDATION TESTS

The validation tests that were described in section 3.1.3 were put together to test the method developed here to generate TFs without producing isomorphic topologies. They also served to describe the current limitations of this approach. However no evidence is provided that these tests are sufficient. Research is required to review these tests, and see what is needed to make it possible to provide guarantees of this or any other method of generating TFs.

4.1.2.2. EXPLORE DIFFERENCE BETWEEN SAT AND CPG

In section 3.4.2.2 it was shown how when retrieving all TPBCs, the CPG significantly outperformed the SAT solver, and its computation time scaled at a shallower rate than that of the SAT solver when increasing the size of the UAMs/set of components. It was also explained in section 3.4.2.2 how that difference might have been caused. Similarly it was emphasised that this does not need to mean that SAT solvers are ill-suited for topology generation. Research is required to find out what the cause is of the large performance differences, and then see whether these performance differences can be eliminated by for example feeding the SAT solver further redundant constraints.

4.1.2.3. CONSTRAINTS FOR SAT SOLVER FOR FEASIBLE TOPOLOGY GENERATION

In section 3.5 the method that was used to develop TFs while avoiding isomorphic topologies was presented. However it remains an open research question to see how this method can be written in the form of constraints that are compatible with SAT solvers.

Both the port-to-port constraint that was used to produce TPBs in section 3.3 and the compatibility constraint that was used to produce TPBCs in section 3.4 are static constraints, i.e. as long as the set of components $\mathcal{S}_c$ is not changed, the constraints always restrict the same ports, no matter the set of requirements $\mathcal{S}_r$ used. The constraints for TFs though would have to work with the requirement chain method and the method that avoids isomorphic topologies in mind, which depend both on the set of components $\mathcal{S}_c$ and the set of requirements $\mathcal{S}_r$.

Still, as was explained in section 3.5.3, TFs are the exception to all of the TPs that exist. And in section 3.4.2.1 it was presented that the more constrained a SAT problem is, the faster a SAT solver can obtain the solutions to the SAT problem. While with the current port-to-port and compatibility constraints, the SAT solver is quite slow to obtain all TPBCs, if the method towards developing TFs without isomorphic topologies is integrated as further constraints, the SAT solver may be able to produce its results significantly faster.

4.1.2.4. ADDING MORE REQUIREMENTS

So far only a single output requirement was considered in order to develop the TFs. In practice several more requirements may be placed on a design. Further contributions are necessary in order for these additional requirements to be taken into account. The requirement chain method would then not be initiated on a single output requirement, but on multiple requirements. This may also lead to faster developments of the TFs.

4.1.2.5. USE MATRICES TO PRODUCE REQUIREMENT CHAINS

As was explained in section 3.5.2, the requirement chain currently implemented is largely based on how the CPG operates, and works by considering each port one by one. However UAMs and boolean composition can be used in order to develop requirement chains as well.

As was explained in section 3.5.2.1 and illustrated in Figure 3.19, first the output requirement is used in order to extract from the local map the relevant inputs u_1 and u_2. This can be done using the boolean composition shown in (4.1).

$$\underbrace{\begin{bmatrix}1 & 0 & 0 & 0 & 0\end{bmatrix}}_{\mu([\text{R-ID}(R_o)])} \circ \underbrace{\begin{bmatrix}1 & 1 & 0 & 0 & 0\\ 1 & 0 & 0 & 0 & 0\\ 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 0 & 1\end{bmatrix}}_{\text{Local map}} = \underbrace{\begin{bmatrix}1 & 1 & 0 & 0 & 0\end{bmatrix}}_{[u_1..u_5]} \tag{4.1}$$

Then, in the same example of section 3.5.2.1, it is looked which outputs can match the inputs u_1 and u_2. It is found that the outputs y_3,y_4 and y_5 are compatible. The same answer can be obtained when the output vector [1,1,0,0,0] from (4.1) is fed into the boolean composition shown in (4.2).

$$\underbrace{\begin{bmatrix}1 & 1 & 0 & 0 & 0\end{bmatrix}}_{[u_1..u_5]} \circ \underbrace{\begin{bmatrix}0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1 & 1\\ 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0\\ 0 & 1 & 0 & 0 & 0\end{bmatrix}}_{\text{Compatibility matrix}} = \underbrace{\begin{bmatrix}0 & 0 & 1 & 1 & 1\end{bmatrix}}_{[y_1..y_5]} \tag{4.2}$$

This process can be iterated to produce the rest of the requirement chain, as is shown in (4.3) and (4.4).

$$\underbrace{\begin{bmatrix}0 & 0 & 1 & 1 & 1\end{bmatrix}}_{[y_1..y_5]} \circ \underbrace{\begin{bmatrix}1 & 1 & 0 & 0 & 0\\ 1 & 0 & 0 & 0 & 0\\ 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 0 & 1\end{bmatrix}}_{\text{Local map}} = \underbrace{\begin{bmatrix}0 & 0 & 1 & 1 & 1\end{bmatrix}}_{[u_1..u_5]} \tag{4.3}$$

$$\underbrace{\begin{bmatrix}0 & 0 & 1 & 1 & 1\end{bmatrix}}_{[u_1..u_5]} \circ \underbrace{\begin{bmatrix}0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1 & 1\\ 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0\\ 0 & 1 & 0 & 0 & 0\end{bmatrix}}_{\text{Compatibility matrix}} = \underbrace{\begin{bmatrix}0 & 1 & 0 & 0 & 0\end{bmatrix}}_{[y_1..y_5]} \tag{4.4}$$

This method of producing requirement chains is significantly more straightforward than the method that was implemented in TopoGen and described in section 3.5.2.2, since it is not necessary to consider ports one by one, but instead the Local map and the Compatibility UAMs are used. Additionally the approach shown here seems also suitable to consider multiple output requirements, which is one of the recommendations to explore further as stated in section 4.1.2.4. However this approach still needs to be worked out further in order to combine it with the method that avoids isomorphic topologies.

4.1.2.6. DEVELOPING TOPOLOGIES ON DIFFERENT SETS

As was explained in section 3.5.4, the method that was currently implemented into TopoGen makes mistakes in the TFs it produces for the ISO-car 2.2BT test. In that same section it was also already explained that to solve this issue, the requirement chain should not be developed just on one set, as it was done with the set `explored_u2ny_connections`, but rather the sets should be split into as many branches that are found each time a branching in the requirement chain is found. This should make it possible to check what instances have already been used for each TF and avoid the mistakes that have been found in the ISO-car 2.2BT test. Putting together this strategy though remains to be done.

BIBLIOGRAPHY

[1] *2050 Long-Term Strategy - European Commission.* URL: https://climate.ec.europa.eu/eu-action/climate-strategies-targets/2050-long-term-strategy_en (visited on 08/08/2024).

[2] Romeo Danielis, Mariangela Scorrano, and Marco Giansoldati. "Decarbonising Transport in Europe: Trends, Goals, Policies and Passenger Car Scenarios". In: *Research in Transportation Economics* (Mar. 2022). URL: https://linkinghub.elsevier.com/retrieve/pii/S0739885921000408 (visited on 08/12/2024).

[3] *Energy Technology Perspectives 2020.* International Energy Agency, 2020. URL: https://www.iea.org/reports/energy-technology-perspectives-2020 (visited on 08/13/2024).

[4] H.K. Fathy et al. "On the Coupling between the Plant and Controller Optimization Problems". In: *Proceedings of the 2001 American Control Conference. (Cat. No.01CH37148).* Proceedings of American Control Conference. Arlington, VA, USA: IEEE, 2001. URL: http://ieeexplore.ieee.org/document/946008/ (visited on 06/10/2024).

[5] Emilia Silvas et al. "Review of Optimization Strategies for System-Level Design in Hybrid Electric Vehicles". In: *IEEE Transactions on Vehicular Technology* (2016). URL: http://ieeexplore.ieee.org/document/7442900/ (visited on 06/07/2024).

[6] E.W.R. van den Belt. "Optimisation Approach of a Hybrid Multi-Axle Heavy-Duty Military Vehicle". Control Systems Technology: TU-Eindhoven, June 3, 2021. URL: research.tue.nl/en/studentTheses/optimisation-approach-of-a-hybrid-multi-axle-heavy-duty-military- (visited on 08/06/2024).

[7] Aart-Jan Kort et al. "Automated Multi-Level Dynamic System Topology Design Synthesis". In: *Vehicles* (Nov. 28, 2020). URL: https://www.mdpi.com/2624-8921/2/4/35 (visited on 06/07/2024).

[8] Emilia Silvas et al. "Functional and Cost-Based Automatic Generator for Hybrid Vehicles Topologies". In: *IEEE/ASME Transactions on Mechatronics* (Aug. 2015). URL: https://ieeexplore.ieee.org/document/7064789 (visited on 06/07/2024).

[9] Hannah Ritchie. "Cars, Planes, Trains: Where Do CO_2 Emissions from Transport Come From?" In: *Our World in Data* (Mar. 18, 2024). URL: https://ourworldindata.org/co2-emissions-from-transport (visited on 08/08/2024).

[10] *Reducing Emissions from the Shipping Sector - European Commission.* URL: https://climate.ec.europa.eu/eu-action/transport/reducing-emissions-shipping-sector_en (visited on 08/13/2024).

[11] Florian Dugast et al. "Topology Generation of Naval Propulsion Architecture". In: *Modelling and Optimisation of Ship Energy Systems 2023* (Jan. 3, 2024). URL: https://proceedings.open.tudelft.nl/moses2023/article/view/667 (visited on 06/18/2024).

[12] *Vehicle Fleets: Service Life and Retirement Age by Vehicle Type?* Thunder Said Energy. URL: https://thundersaidenergy.com/downloads/vehicle-fleets-service-life-and-retirement-age-by-vehicle-type/ (visited on 08/08/2024).

[13] William Spitz et al. *Development Cycle Time Simulation for Civil Aircraft*. NASA, 2001.

[14] *Airworthiness Certification | Federal Aviation Administration*. Jan. 6, 2023. URL: https://www.faa.gov/aircraft/air_cert/airworthiness_certification (visited on 08/12/2024).

[15] *Timeline for Aircraft Carrier Service*. In: *Wikipedia*. June 9, 2024. URL: https://en.wikipedia.org/w/index.php?title=Timeline_for_aircraft_carrier_service&oldid=1228125437 (visited on 08/12/2024).

[16] Mehmet Cem Demirci. *Why Does an Aircraft Carrier Take so Many Years to Build?* May 23, 2021. URL: https://navalpost.com/why-does-an-aircraft-carrier-take-so-many-years-to-build/ (visited on 08/12/2024).

[17] Steven Wilkins et al. "Co-Design and Energy Management for Future Vessels". In: *Modelling and Optimisation of Ship Energy Systems 2023* (2023). URL: https://proceedings.open.tudelft.nl/moses2023/article/view/666 (visited on 07/12/2024).

[18] Emmanuel Vinot et al. "HEVs Comparison and Components Sizing Using Dynamic Programming". In: *2007 IEEE Vehicle Power and Propulsion Conference*. 2007 IEEE Vehicle Power and Propulsion Conference (VPPC). Arlington, TX, USA: IEEE, Sept. 2007. URL: http://ieeexplore.ieee.org/document/4544143/ (visited on 09/29/2023).

[19] Anne Helmenstine. *How Many Atoms Are in the World?* Science Notes and Projects. May 10, 2022. URL: https://sciencenotes.org/how-many-atoms-are-in-the-world/ (visited on 06/07/2024).

[20] Gioele Zardini et al. *Task-Driven Modular Co-design of Vehicle Control Systems*. Sept. 20, 2022. arXiv: 2203.16640 [cs, eess]. URL: http://arxiv.org/abs/2203.16640 (visited on 06/18/2024). Pre-published.

[21] Gioele Zardini et al. "Co-Design of Embodied Intelligence: A Structured Approach". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 27, 2021. arXiv: 2011.10756 [cs, eess]. URL: http://arxiv.org/abs/2011.10756 (visited on 06/18/2024).

[22] Andrea Censi. *A Mathematical Theory of Co-Design*. Oct. 12, 2016. arXiv: 1512.08055 [cs, math]. URL: http://arxiv.org/abs/1512.08055 (visited on 06/18/2024). Pre-published.

[23] Mark E. J. Newman. *Networks: An Introduction*. Reprinted. Oxford: Oxford University Press, 2016.

[24] Murat Arcak, Chris Meissen, and Andrew Packard. *Networks of Dissipative Systems*. SpringerBriefs in Electrical and Computer Engineering. Cham: Springer International Publishing, 2016. URL: http://link.springer.com/10.1007/978-3-319-29928-0 (visited on 06/07/2024).

[25] Mathworks. *Basic Principles of Modeling Physical Networks - MATLAB & Simulink*. 2023. URL: https://www.mathworks.com/help/simscape/ug/basic-principles-of-modeling-physical-networks.html (visited on 08/16/2023).

[26] Sudhir Jonathan. *Understanding UUIDs, ULIDs and String Representations*. URL: https://sudhir.io/uuids-ulids (visited on 06/21/2024).

[27] Gioele Zardini, Andrea Censi, and Emilio Frazzoli. "Co-Design of Autonomous Systems: From Hardware Selection to Control Synthesis". In: *2021 European Control Conference (ECC)*. June 29, 2021. arXiv: 2011.10758 [cs, eess]. URL: http://arxiv.org/abs/2011.10758 (visited on 06/07/2024).

[28] George J. Klir and Bo Yuan. *Fuzzy Sets and Fuzzy Logic: Theory and Applications*. Upper Saddle River, N.J: Prentice Hall PTR, 1995.

[29] Stephen D. Brown et al. *Fundamentals of Digital Logic with VHDL Design*. 3. ed. [Intern. student ed.] McGraw-Hill Series in Electrical and Computer Engineering. Boston: McGraw-Hill Higher Education, 2009.

[30] Goupil. *Goupil G4 Datasheet*. URL: https://www.goupil-ev.com/en/g4 (visited on 03/19/2024).

[31] *Universally Unique Lexicographically Sortable Identifier (ULID) in Python 3*. June 25, 2024. URL: https://github.com/ahawker/ulid (visited on 06/27/2024).

[32] Oxford Maths. *Number of Atoms in the Universe*. Oxford Education Blog. Nov. 24, 2015. URL: https://educationblog.oup.com/secondary/maths/numbers-of-atoms-in-the-universe (visited on 06/28/2024).

[33] Rod Pierce. *Combinations and Permutations*. Math is fun. URL: https://www.mathsisfun.com/combinatorics/combinations-permutations.html (visited on 06/07/2024).

[34] *A Peek Inside SAT Solvers*. Dec. 2, 2016. URL: https://www.youtube.com/watch?v=d76e4hV1iJY (visited on 07/02/2024).

[35] Prince, S and Srinivasa, C. *Tutorial #10: SAT Solvers II: Algorithms*. Borealis AI. URL: https://www.borealisai.com/research-blogs/tutorial-10-sat-solvers-ii-algorithms/ (visited on 07/02/2024).

[36] GeekforGeeks. *Conflict Driven Clause Learning (CDCL)*. GeeksforGeeks. Nov. 16, 2022. URL: https://www.geeksforgeeks.org/conflict-driven-clause-learning-cdcl/ (visited on 01/12/2024).

[37] Verification and Synthesis, director. *Tutorial / SAT for Problem Solving*. Nov. 22, 2021. URL: https://www.youtube.com/watch?v=GdrI3IESyVY (visited on 01/03/2024).

[38] MicrosoftResearch. *Z3Prover*. GitHub. URL: https://github.com/Z3Prover/z3/wiki/Home (visited on 07/02/2024).

[39] D.W. and Richard. *Answer to "K-out-of-N Constraint in Z3Py"*. Stack Overflow. Mar. 28, 2017. URL: https://stackoverflow.com/a/43081930/3604362 (visited on 07/02/2024).

[40] Tim Peters. *Answer to "Itertools.Product Eliminating Repeated Elements"*. Stack Overflow. Nov. 2, 2013. URL: https://stackoverflow.com/a/19744905/3604362 (visited on 07/03/2024).

[41] Rod Pierce. *Injective, Surjective and Bijective*. Math is fun. URL: https://www.mathsisfun.com/sets/injective-surjective-bijective.html (visited on 07/01/2024).

[42] Wikipedia. *Stirling's Approximation*. URL: https://en.wikipedia.org/wiki/Stirling%27s_approximation (visited on 07/01/2024).

[43] Matt Ginsberg. *Pseudo-Boolean and Cardinality Constraints*. Feb. 19, 2004. URL: https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume21/dixon04a-html/node14.html (visited on 07/03/2024).

A

IMPORTANT DEFINITIONS

A.1. FUZZY COMPOSITION

In [28] (section 5.3, Binary Fuzzy Relations) it is described how relations can be composed. Consider two fuzzy relations $P(X,Y)$ and $Q(Y,Z)$, with the common set Y. Then a fuzzy composition allows one to obtain a relation $R(X,Z)$.

The standard composition of these relations, denoted as $P(X,Y) \circ Q(Y,Z)$, is defined as shown below:

$$R(x,z) = [P \circ Q](x,z) = \max_{y \in Y} \min[P(x,y), Q(y,z)]$$

for all $x \in X$ and all $z \in Z$. This composition is also referred as the max-min composition.

On page 126, next to equation 5.8, an example of such a composition is provided. There it made clear that the max-min composition works like a regular matrix multiplication, but the product and sum operations are replaced with min and max operations respectively. Therefore the max-min composition can be written in matrix-index notation like shown below:

$$r_{i,j} = \max_{k \in K} \min(p_{i,k}, q_{k,j})$$

A.2. COMBINATIONS & PERMUTATIONS

In [33] provides an explanation into what combinations and permutations are.

Consider as an example that there is a set of $n = 3$ elements $S := \{0,1,2\}$. If one is to draw $r = 2$ times (without repetition), then two different permutations could be $p_1 = (0,1)$ and $p_2 = (1,0)$, even though it represents the one and the same combination, i.e. $c = p_1 = p_2$.

As shown by the example above, in combinations the order of the elements drawn does not matter, while for permutations it does. Additionally it is important to specify whether the draws are made with or without repetition. From the example mentioned above, with repetition, a possibly permutation/combination could have been $(2,2)$.

In [33] the table shown in Table A.1 is provided in order to calculate the number of combinations or permutations when considering n options, r draws, and whether the same option can be drawn multiple times (a.k.a. with or without repetition).

	with repetition	**no repetition**
Permutations	n^r	$\frac{n!}{(n-r)!}$
Combinations	$\frac{(r+n-1)!}{r!(n-1)!}$	$\frac{n!}{r!(n-r)!}$

Table A.1: Formulas to calculate the number of combinations or permutations with r draws and n options.

A.3. INJECTIVE, SURJECTIVE AND BIJECTIVE FUNCTIONS

As is explained in [41], a function describes how a set A is mapped onto a set B, i.e. $f: A \mapsto B$. However functions can be differentiated on the type of mapping that is performed. In Figure A.1, provided by [41], an overview of the different types is shown.

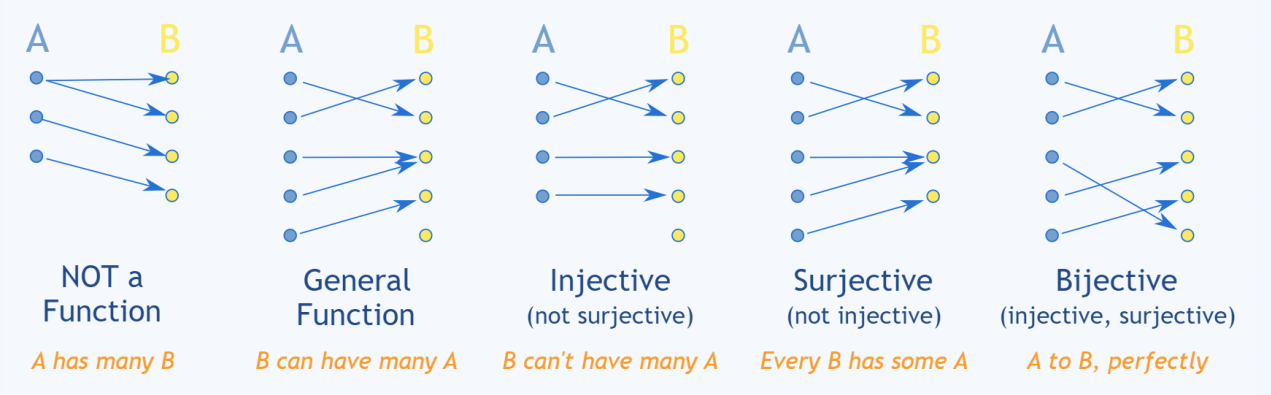


Figure A.1: Different types of functions, differentiated by the type of mapping.

The different types of functions can be differentiated as follows:

- **Not a function:** A function can never map one element to many. For example $f(x) = 7 \vee 9$ is not a function [41].
- **General function:** Each element in A is mapped to some element in B. It is possible that multiple elements from A point to the same element in B.
- **Injective function:** Unlike the general function, with an injective function, it is not possible that multiple elements from A point to the same element in B.
- **Surjective function:** Unlike the general function, with a surjective function, all elements of B are being mapped to from elements in A.
- **Bijective function:** A bijective function is both injective and surjective, i.e. all elements of A map to different but all elements of B.

A.4. STIRLING'S APPROXIMATION

As explained in [42], Stirling's approximation is a popular approximation of the factorial $n!$. Below the approximation:

$$\ln(n!) = n\ln(n) - n + \mathcal{O}(\ln(n))$$

where $\mathcal{O}$ indicates the error terms in the big-o-notation.

The error term itself can be expressed more precisely as $\frac{1}{2}\log(2\pi n) + \mathcal{O}(\frac{1}{n})$, which leads to another more precise approximation of the factorial:

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

where the symbol $\sim$ indicates that the two quantities are asymptotic, i.e. their ratio tends to 1 as n goes to infinity.

The following bounds hold for all $n \geq 1$:

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\frac{1}{12n+1}} \leq n! \leq \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\frac{1}{12n}}$$

A.5. CARDINALITY AND PSEUDO-BOOLEAN CONSTRAINTS

Consider the clause shown below, provided as an example by [43]:

$$x \vee y \vee \neg z$$

As [43] explains, this can be expressed as a linear equation like:

$$x + y + \bar{z} \geq 1$$

where x, y and z are variables that can take the values 1 or 0, and $\bar{z} := (1 - z)$. This leads to a at-least-1-out-of-n constraint, where n is the number of variables considered.

In [43] it is further generalised to a at-least-k-out-of-n constraints, leading to two forms:

1. A cardinality constraint:
$$\sum_i l_i \geq k$$
where $l_i \in \{0, 1\}$ and k is an integer.

2. A pseudo-boolean constraint:
$$\sum_i w_i l_i \geq k$$
where $l_i \in \{0, 1\}$, and k and w_i are integers. Unlike the Cardinality constraint the Pseudo-boolean form can be used to express other form of constraints, like the example shown in [43], $2a + b + \bar{c} \geq 2$.