An Abductive Approach to Design Structure Matrix (DSM) Partitioning using Frequency Domain Scoring

by

Jonathan Ho Jun

Bachelor of Engineering (EEE)
Nanyang Technological University, 2010

Submitted to the System Design and Management Program
in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Engineering and Management

at the

Massachusetts Institute of Technology

February 2018

Signature redacted

Signature of Author _____

Jonathan Ho Jun
System Design and Management Program
January 4, 2018

Signature redacted

Certified by _____

Bruce Cameron
Director, System Architecture Lab
Thesis Supervisor

Signature redacted

Accepted by _____

Joan Rubin
Executive Director, System Design and Management Program

This page is intentionally left blank.

An Abductive Approach to Design Structure Matrix (DSM) Partitioning using Frequency Domain Scoring

by

Jonathan Ho Jun

Submitted to the System Design and Management Program on January 4, 2018
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Engineering and Management

Abstract

A key benefit of the DSM representation is that it gives a visual interpretation of relationships between its elements. The array format allows us to sort the elements using clustering algorithms which try to group the relationships into modules which are as independent as possible. There are a number of clustering algorithms available which may each end up sorting the DSMs differently using different objectives, for example, activities in a time-based DSM can be sequenced to reduce iterations or to improve concurrency. However, most of these algorithms take a deductive approach which results in only one 'optimal' output. If an abductive approach is used instead, multiple solutions can be generated for the user to evaluate, some which may provide insight on useful configurations that he or she may have overlooked.

In electrical engineering, we often make use of transforms to convert time domain signals into frequency domain signals in order to glean additional information which may not have been initially apparent. In this respect, using a frequency domain transform on a DSM matrix gives us additional insights into the relationships represented. An example of one such insight would be into the sorted-ness of a DSM to which module cuts can be defined. By applying a frequency transform to a pixel representation of the DSM and examining the transform coefficients, we gain an understanding of what image patterns exist in the DSM. Rules pertaining to these coefficients could then be defined which would classify a DSM as well sorted (with the dependencies being grouped up) or being unsorted (with the dependencies being scattered).

This thesis demonstrates the above technique to rank each permutation of an 8x8 matrix on their conformance to certain rules or behaviors in order to filter out useful configurations in an abductive approach. When comparing the highest-ranking hypotheses against the optimal result from other clustering and sequencing algorithms, this algorithm performed on par with them to reduce external dependencies and iterations respectively. The frequency based scoring was also shown to be a useful metric when determining the optimal module cut of a system.

Thesis Supervisor: Bruce Cameron
Title: Director, System Architecture Lab

This page is intentionally left blank.

Acknowledgements

There are many people I would like to thank for being a part of my life during this amazing journey which culminated in this thesis and degree. I felt that I have grown during my time here at MIT, and each and every one of these people had a stake in it.

First and foremost, I must thank my advisor, Bruce Cameron, without whom this thesis would not have been possible. Despite his busy schedule, he took the time to conscientiously guide me during the entire thesis process. His valuable insight and experience in the domain of system architecture helped me to grow my thesis idea in ways that I would never have imagined.

I would like to thank Joan Rubin, the executive director of the SDM course as well as the other members of the SDM staff for their support over the last 16 months, the course could not have been run any better. I am especially appreciative to Joan for helping me to secure a software license to Lattix on short notice which contributed a great deal to this thesis.

I would like to thank my company, Defence Science & Technology Agency (DSTA), for affording me this opportunity to pursue my postgraduate studies. I would like to specially mention my supervisor, Chee Hoong, and my director, Hee Tiam, for being outstanding mentors and supporting my endeavor to develop myself professionally and personally in this regard.

I would like to thank my friends here in MIT for enriching my life with plenty of entertainment and immersing me in a culture of diversity and brilliance. I had fun, and I hope you all did too.

Last but not least, I would like to thank my family, both in US and abroad for the care they provided- My parents, Lick and Maggie, for constantly checking in on me and making sure I was eating right, as well as providing moral support and advice. My aunt Susan and uncle Jeff for inviting me over to share their place during the holidays and always making sure I had everything I needed. Most of all, my fiancée, Elsa, who put up with me being away for 16 months, but still always being there to encourage me.

This page is intentionally left blank.

Table of Contents

List of Figures

List of Tables

Table of Abbreviations

DCT	Discrete Cosine Transform
DSM	Design Structure Matrix

1. Introduction

In recent decades, systems have gotten increasingly complex. An example of this could be seen in the automobile industry where cars are being transformed into complex electronic machines from previously simple mechanical machines. Sensors, controllers and actuators have been added to create new features such as passive entry door locking, autonomous driving as well as interfacing to other smart devices[1]. With the rapid increase in number of components and complexity of such systems, managing them via traditional means has become increasingly difficult as well.

Baccarini [2] proposed that the definition of complexity is 'consisting of many varied interrelated parts' and can be operationalized in terms of differentiation and interdependency. He differentiates complexity from two other characteristics of size and uncertainty.

A common feature of complex systems is the presence of emergence: these are characterized by properties of a system which are not apparent from its components in isolation but which result from the interactions they form when placed together in a system. These emergent properties may have beneficial or detrimental consequences to the function of the system. The payoff for using a complex system can be tremendous if there is a net benefit drawn from the system's emergent properties.

Complexity can be costly in terms of having a much larger number of components and interactions to manage, requiring more awareness and effort on the part of the architect. This can be to the point where the awareness required exceeds the limitations of a human and has to be somehow made simpler to perceive. By separating a system into hierarchical, independent modules, and making decisions using competing criteria (such as the size of each module or the configuration of interfaces) addresses human limitations in dealing with complexity. An example of this in practice would be companies dividing their work into departments that each deal with different issues which they specialize in. [3][4]

A tradeoff thus exists between the cost of managing complexity in a system versus reaping the emergent functionality from its varied interrelated parts [3]. This creates the impetus to use complex systems, whilst developing methods and tools to manage complexity and keep it under control.

2. Modularization, and its ties to managing complexity

According to Baldwin and Clark [5] , modularization serves three purposes:

To manage complexity
The system can be decomposed into its modules and managed separately, as long as the interfaces are handled properly. Viewing the system in these bite-sized chunks allows a

complex system to be more easily understood and worked on. This division of cognitive labor also makes possible the graceful evolution of knowledge about the system [6].

To allow work to be done in parallel
Work on different modules generally can be performed simultaneously, which allows for shorter start-to-finish times.

To accommodate future uncertainty
A prime example of this would be in obsolescence management. In creating modules within a system, obsolete elements can be easily substituted or changed out with little cost without changing the functionality of the system. The system can be said to be 'tolerant to uncertainty' as options can be made for each individual module rather than the system as a whole.

Modularity unfortunately has a cost as well, which may offset the value it creates. On top of the additional effort and time required to make decisions, run tests and find ideal points to segment a system into modules, bringing modularity to a system may introduce inefficiencies which may impact performance or increase costs as well (for example, by adding redundancies between modules). It is thus also the role of a system architect to make decisions as to how best to balance these positives and negatives.

We will particularly be looking at a tool called The DSM (Design Structure Matrix) commonly employed by system architects. This is a simple, albeit powerful tool which can be used to map the dependencies of activities in a process (in a time-based DSM), or the relationships between the decomposed elements of a system (In a static DSM)[7] in the form of a square N x N matrix[8].

	A	B	C	D	E	F	G	H	I
A	.	1			1				1
B		.					1		1
C			.					1	
D				.	1	1	1		
E	1			1	.		1		
F					1	.	1		
G		1		1	1	1	.		
H			1					.	1
I	1	1						1	.

Figure 1 - Showing sample DSM matrix for elements A to I

The above DSM represents a system consisting of 9 elements, labeled A-I. The relationships or dependencies between the various elements are represented by '1's on the DSM (for example, element E has a dependency on element A, and element A has a dependency on element E, so there are indications where E intersects A and A intersects E).

By mapping these elements in a process or physical system out in a DSM, the order for which these activities or elements in the DSM can be rearranged to achieve certain objectives such as introducing modularity (by grouping dependencies into clusters along the center diagonal). For example, in the above DSM, merely changing the positions of element 'C' with element 'I' in the DSM representation causes the dependencies to re-map themselves and aggregate closer around the center diagonal:

	A	B	I	D	E	F	G	H	C
A	.	1	1		1				
B		.	1				1		
I	1	1	.						
D				.	1	1	1		
E	1			1	.		1		
F					1	.	1		
G		1		1	1	1	.		
H								.	1
C								1	.

Figure 2-Showing the sample DSM rearranged and clustered into modules

By changing the position of element 'C' and element 'I' from figure 1, highly interdependent elements are moved adjacent to each other on the DSM allowing for drastic improvements to the system in its entirety[8]. Grouping these highly interdependent elements into a modular unit (as denoted by the red boxes in figure 2) allows for the system's complexity to be better managed in a hierarchal way. Take for instance a department in an organization constantly having to interact with another department- if the two departments were to be combined into one, there would be greater efficiency for the organization as a whole from the elimination of inefficient interfaces between the two groups of people.

This reordering of elements in a DSM to form modules is known as 'clustering' and is commonly applied to architectures represented in DSM format.

3. Clustering Algorithms

Earlier we mentioned 2 types of DSMs, time-based DSMs representing process architecture and static DSMs representing system architecture. There are different kinds of algorithms which can be applied to both types of DSM.

In order to achieve modularity in a static DSM, the components can be rearranged and partitioned into groups. Algorithms which perform the rearrangement of these types of DSMs are known as clustering algorithms and come in many forms in order to fulfil their functions. For example, some clustering algorithms may reorder a system architectural DSM to optimize module size based on their internal and external dependencies, or work towards reducing costs between interfaces.

Baldwin and Clark 2004[5] demonstrate the modularization of a laptop system by design rationalization: setting specific design rules such as the co-location of 2 different components within the system. These design rules can then be represented in a DSM as a new 'design rules' block, which acts as a bridge between the components affected, allowing dependencies to be drawn to this block instead of having the dependencies drawn directly between the components. The advantage of this clustering algorithm is its versality; it can be used on practically any system. However, this involves additional extra steps in the design process involving identifying dependencies, defining and encapsulating existing modules, creating and maintaining a set of design rules as part of the system and establishing additional system integration as well as verification and validation in order to assemble and test for emergent problems between the modules.

Whitfield, Smith, and Duffy 2002 [9] make use of a clustering criterion to first identify modules within the system. A Module Strength Indicator (MSI) is then used to score various system configurations in order to determine the strength of their modularity. This score takes into account 2 equations, one for internal dependencies and one for external dependencies, subtracting them to get an overall score. For example, if there are only maximum weight internal dependencies (score of 1) without any external dependencies (score of 0) the overall system would be said to have a score of 1-0 = 1 for its MSI. This MSI is calculated for all possible modules in the system. The MSI scores are then plotted on another matrix showing the various possible degrees of modularity and their respective modules, allowing the architect to visualize the modularity weights when varying the modules. This clustering algorithm boasts good visualization of modules vs modularity, as well as modular hierarchy. This visualization allows the user to quickly eliminate many non-modular configurations for a given DSM configuration. One point to note about this algorithm is that it does not rearrange the DSM to achieve a better modular configuration, rather, it acts as a visual aid for the architect to choose an optimal modular configuration for a given DSM configuration only.

Warfield. J [10] clusters by forming partitions containing the smallest possible square submatrices along the center diagonal. This algorithm is straightforward but there is some pre-

arrangement which needs to be done prior to the partitioning and different permutations from this pre-arrangement would result in different partitions being formed. Ideally, the pre-arrangement should strive to have all the ones in the matrix close to the center diagonal such that they become part of a partition cluster, or on the bottom left of the center diagonal so that a block triangular matrix can be formed when the partitions are defined.

Thebeau[11] takes a stochastic hill-climbing approach to clustering based off work from Idicula[12] and Gutierrez[13]. The algorithm uses a matrix to keep track of the association between elements and clusters. It randomly moves elements from one cluster to another to see if there is an improvement using a score defined for each cluster and overall system. The goal is to minimize the total interactions between clusters while maximizing the total interactions within clusters. Due to the stochastic nature of this algorithm, it needs to be run thousands of times in order to produce a useful result, this leads to very long run times for larger DSMs.

Algorithms which reorder time-based DSMs are known as sequencing algorithms as they reorder the sequence of activities in a process. Unlike clustering algorithms used on static DSMs which may have very different goals from algorithm to algorithm, sequencing algorithms share a similar goal of reducing iterations in a time-based DSM, the differences normally lie in how the different algorithms identify loops of coupled elements[14]. Below are a few examples of sequencing algorithms.

Cook[15] uses a genetic algorithm to perform sequencing in a time-based DSM using a more specialized 'divide and hybridize' method to minimize iteration while maximizing concurrency. This algorithm divides the population into smaller parts and optimizes each of these parts separately. These smaller parts are then hybridized to find a solution between the various optima. This algorithm works better on DSMs with higher coupling densities (as DSMs with lower coupling densities are more complex problems for genetic algorithms to solve and would require a larger population size).

S.Cho[16] performs sequencing by first grouping strongly connected components together, then creating ordered layers such that connections to any elements in layers above are minimized. The algorithm then arranges the layers such that each component is in the lowest layer possible, ensuring that as few dependencies as possible are above the center diagonal. Limitations of this algorithm are that it does not include any process optimization technique such as finding optimal resource allocation to minimize lead time. This algorithm also makes an assumption that the time for each of the tasks are independent of each other, which may not be true when used in certain situations.

From the above examples, we can see that there are a multitude of approaches to clustering and sequencing, which result in different optimums being found. These various clustering algorithms, when applied to the same DSM matrix, may lead to different insights and solutions as to how to modularize a system. For example, an algorithm looking to reduce interfaces may work to reduce the number of dependencies between modules by favoring large module sizes, while another algorithm which uses a bridge like Baldwin and Clark [5] may define a interface

'bus' for the system resulting in an entirely different configuration altogether for the same system.

4. Motivation

According to Ho[17], There are three modes of reasoning found in literature: induction, abduction and deduction. Induction involves using a result and a case to ascertain a rule (input), abduction involves determining the case from the rule and output while deduction establishes the result based on the case and rule.

The following table demonstrates the difference between deductive, inductive and abductive reasoning:

Mode	Example
Deductive reasoning	If Jack exercises he gets sweaty. Jack just exercised, therefore he is sweaty. *Rule: If Jack exercises he gets sweaty.* *Case: Jack just exercised,* *Result: Therefore, he is sweaty.*
Inductive reasoning	John studies a lot. He has no grey hairs. Therefore, studying does not give people grey hairs. *Case: John studies a lot.* *Result: He has no grey hairs.* *Rule: Therefore, studying does not give people grey hairs.*
Abductive reasoning	People who like animals own a pet. Mary has a dog. Therefore, Mary likes animals. *Rule: People who like animals own a pet.* *Result: Mary has a dog.* *Case: Therefore, Mary likes animals.*

Table 1-Showing examples of the different modes of reasoning

Note that in the abductive reasoning example above, Mary owning a dog may not be solely due to the fact that Mary likes animals; it is only a hypothesis. Mary could own a dog for other reasons for which the rules have not been defined or established yet. If another rule exists that 'blind people own a dog', there may now be another hypothesis as to why Mary owns a dog- she may be blind. The format of abductive reasoning is characterized as such:

"A phenomenon Z is observed.

Among hypotheses A, B and C, A and B are capable of explaining Z, therefore there is reason to pursue hypotheses A and B."

Deduction, like induction is characterized as symbolic logic, unlike abduction which is instead synonymous with critical thinking[17]. Abduction, unlike deduction and induction, is excellent for hypothesis generation.

One commonality that can be seen present in many of the clustering algorithms in literature are that they utilize bottom-up approaches, whereby the process of rearrangement leads to a single result. This is very much akin to the process of deduction: "I have the original configuration of my system and follow a set of fixed steps to cluster and obtain a final configuration of my system". Whitfield [9] comes close to an abductive approach, but stops short of creating set of modular configurations (hypotheses), instead requiring the user to make abductive comparisons from its output.

With abductive thinking, a system architect is capable of generating multiple hypotheses or possible system architecture configurations. As there are many considerations in an architect's decision making, having multiple hypotheses, each leading to a different system configuration could give the architect the edge needed to make certain decisions which may otherwise have been overlooked if only the 'optimal' solution had been presented to him or her.

Taking a deductive approach also makes it difficult for an algorithm to learn or update itself as it works on various DSMs. This is because the process of rearrangement is always bound by a fixed procedure or set of rules. Approaching clustering abductively, or from a top-down perspective would allow the process to evolve or change as more DSMs are worked on by the algorithm. This is especially useful in this era of technology where data, data storage and computational processing are widely accessible leading to the increasing prominence of machine learning.

An abductive approach does have its shortcomings as well. If a large number of hypotheses exists, it may be detrimental to the architect as it may overwhelm him or her. This is why it is crucial to devise a means to pick out the more useful hypotheses for the architect's consideration while discarding the less useful ones.

The fundamental motivation behind this paper is to explore the possibility of taking an abductive rather than a deductive approach to clustering in order to obtain multiple hypotheses or possible modular configurations, and rank them in terms of their usefulness to improve the way in which we modularize systems.

5. Approach

There are multiple ways to implement abductive clustering in an algorithm, as long as we define the case (input matrix to be configured) and a result (a more modular configuration) that we are trying to achieve for the system.

An example of abductive reasoning implemented in algorithmic form is the use of a Hidden Markov Model[18]. In the Hidden Markov Model, all possible hypotheses are generated and assigned a probability or score based on their similarity to a certain pattern; the more likely a hypothesis is to a certain pattern, the higher the score. From this set of scores, the highest is picked and is chosen as the valid hypothesis.

Abduction is also employed in Bayes' theorem, which makes use of a prior and likelihood function to produce a posterior probability distribution, which in turn shows the distribution of the uncertain quantity given the data. In this case, the prior acts as the result, the likelihood function the rule, and the posterior probability distribution acts as the case, of which there are many 'hypotheses' found in the distribution.

If we were to implement an abductive modularization algorithm, the architecture of it would require something similar. Building such an algorithm would require an objective pattern and a scoring system to compare all possible hypotheses against it.

Herein is where the difficulty lies. How to we create or define a pattern to describe a modular system if every system is different? A modularized configuration of system A may look totally different from a modularized configuration of system B simply because the relationships defined in each system are not correlated to each other. If I need to build a robust means to identify patterns which could be used to score any system input into the algorithm, how would we go about doing it?

In electrical engineering, we often make use of transforms to convert time domain signals into frequency domain signals in order to glean additional information which may not have been initially apparent. In this respect, using a frequency domain transform on a DSM matrix may give us additional insights into the patterns of the relationships represented.

We will be using a frequency transform for two reasons:

Firstly, by creating a scoring system in the frequency domain, we de-correlate the pixel positions of the relationships on a DSM for scoring. As every DSM representing various systems are different, the absolute position of each individual relationship is not important, rather what is needed is to look at underlying behaviors and patterns within the DSM when the pixels are placed relative to each other. De-correlating the pixel positions transfers the importance of having a specific cell filled all the time to having a specific pattern to denote modularity. To illustrate, let us look at the following DSMs:

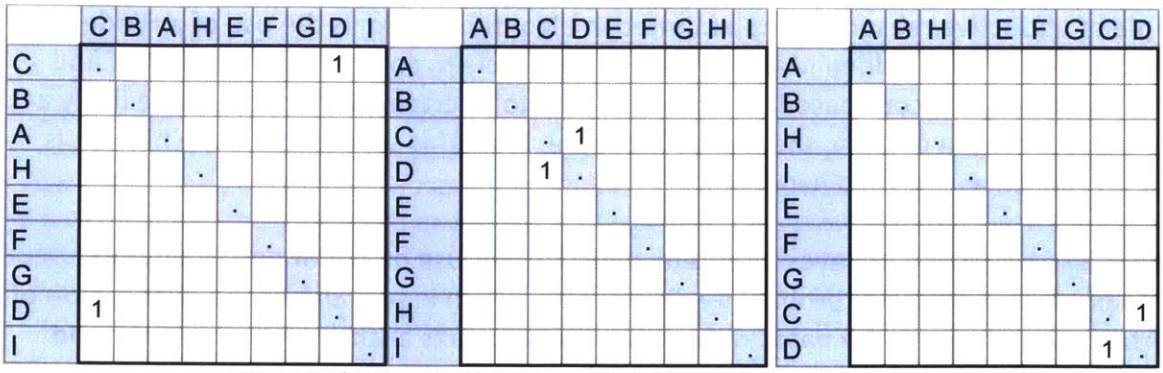


Figure 3- Showing 3 configurations of a same system (by reordering the elements)

All three of these DSM represent the same system but with their elements arranged in different configurations. We can argue that placing elements C and D together (as seen in the middle and right configurations of the DSM) begets a more modular configuration as they can be grouped together into a module (due to their interdependencies). However, it does not matter where in the DSM these two elements are placed as long as they are placed together; it can be said that the configuration in the middle and right DSMs are equally modular. Using a frequency transform to de-correlate the pixel positions of the relationships removes the need to have the elements C and D located at specific positions on the DSM and allow for us to score the system's modularity based solely on patterns we define.

This brings us to the second reason for using a frequency transform. We need to identify patterns to exploit which characterize modularity. These behaviors or patterns are viewed in the time domain but can be more easily defined in the frequency domain which we will talk about more in the next section.

Any frequency transform can suit our purpose; however, we will be making use of one particular transform called the Discrete Cosine Transform. This transform, which is typically employed in image processing is especially synergistic with the visual, pixel-like representation of the DSM.

6. The DCT (Discrete Cosine Transform)

Definition and Principles

Before we delve into the clustering algorithm proper, some explanations pertaining to the methodology is required. The DCT is a transform widely used by the image and video processing domain to compress image data. Compression of the images are made possible due to the fact that humans are not sensitive to small deviations in neighboring pixels, and it is thus possible to remove certain high frequency components with little noticeable change in the image.

Before filtering of these high frequency components in image processing however, the image has to first be transformed into a weighted sum of its frequency components, and that is where the DCT is employed.

The 1-dimensional DCT is characterized by the following equations[19]:

$$C(u) = \alpha(u)\sum_{x=0}^{N-1} f(x)\cos\left[\frac{\pi(2x+1)u}{2N}\right]$$

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{N}} & for \quad u = 0 \\ \sqrt{\frac{2}{N}} & for \quad u \neq 0. \end{cases}$$

for u= 0,1,2,3, ... N-1

while the 2-dimensional DCT simply involves performing this same transform along 2 axes:

$$C(u,v) = \alpha(u)\alpha(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\cos\left[\frac{\pi(2x+1)u}{2N}\right]\cos\left[\frac{\pi(2y+1)v}{2N}\right]$$

for u, v = 0,1,2,3, ... N-1

Performing a 2-dimensional DCT on a NxN matrix yields another NxN matrix indicating the weights of its basis frequency components (shown in the figure below). As we move from the top left downward or to the right, they represent basis frequency components of increasing frequency (As u can see by the increasing frequency of black and white bands which represent the amplitudes). This table constitutes the various patterns which make up the time domain input.

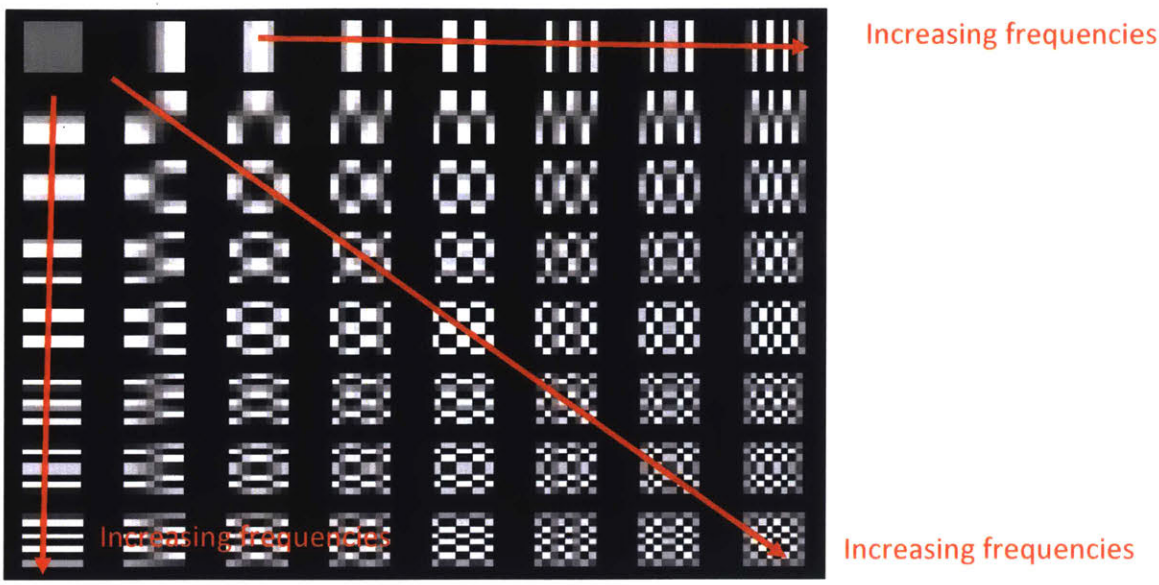


Figure 4- Two dimensional DCT basis functions (N=8). White indicates positive amplitudes, black indicates negative amplitudes and grey indicates zero. [19]

The basis function on the top left is known as the Direct Current (DC) basis frequency component which has zero frequency. Moving each step from left to right of it there would be an increase in horizontal frequency by half a cycle, while moving each step from top to bottom would cause an increase in vertical frequency by half a cycle. The diagonals correspond to a mix between the respective vertical and horizontal frequencies for that respective row and column.

To demonstrate, In the figure below, we have an 8x8 matrix arbitrarily populated with ones and zeros:

Input

1	1	0	1	0	0	0	0
1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0
1	0	1	1	0	0	0	0
0	0	0	0	1	1	0	0
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	0
0	0	0	0	0	0	0	1

Figure 5- Showing 8x8 matrix before DCT

Performing a 2-dimensional DCT transform on the above 8x8 matrix yields another 8x8 matrix shown below:

Transform

2.6250	0.7706	-0.0676	0.1118	-0.1250	0.5622	0.1633	-0.5149
0.8716	2.2234	-0.0876	-0.6054	0.5249	0.2991	-0.3017	-0.3672
-0.1633	0.0106	0.6919	-0.3654	0.1633	-0.1628	-0.4634	-0.3999
-0.2200	-0.8848	-0.2408	1.1112	-0.5139	-0.1082	-0.3247	-0.1551
-0.3750	-0.0609	-0.0676	-0.0833	0.8750	-0.4186	0.1633	0.0407
0.0656	0.1551	0.1169	0.0686	-0.1308	0.7121	-0.1019	-0.1348
-0.0676	0.0044	0.2866	-0.1514	0.0676	-0.0674	-0.1919	-0.1656
-0.0069	-0.1904	-0.2529	0.4509	-0.0758	-0.6054	-0.1576	-0.0466

Figure 6- Showing 8x8 matrix after DCT

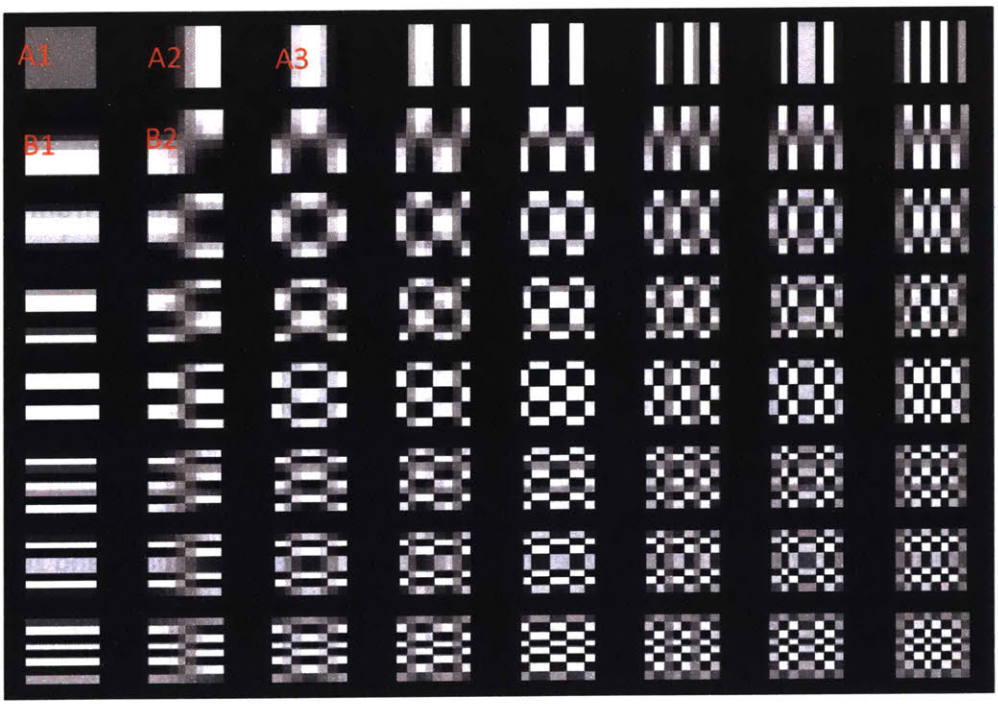


Figure 7-Showing DCT basis functions

The values found in the second 'transformed' matrix indicate the weights of the DCT basis functions required to recreate the input; with respect to the basis function figure above, summing up

2.625*A1+0.7706*A2+(-0.0676) *A3+...+ 0.8716*B1+ 2.2234*B2 +...

so on and so forth for all 64 basis functions and coefficients, we will get back the original input prior to the transform.

Below we see some more examples of 8x8 matrixes and their respective transforms:

Input							
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1

Transform							
8.0000	0.0000	-0.0000	0.0000	0.0000	0.0000	-0.0000	-0.0000
0.0000	0.0000	0.0000	-0.0000	0.0000	0.0000	-0.0000	-0.0000
-0.0000	0.0000	0.0000	0.0000	-0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000	-0.0000	-0.0000	0.0000	0.0000
0.0000	-0.0000	0.0000	-0.0000	0.0000	-0.0000	-0.0000	0.0000
0.0000	-0.0000	-0.0000	-0.0000	-0.0000	0.0000	-0.0000	-0.0000
-0.0000	-0.0000	0.0000	0.0000	0.0000	-0.0000	-0.0000	0.0000
-0.0000	0.0000	0.0000	0.0000	0.0000	-0.0000	0.0000	0.0000

Input							
1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1

Transform							
4.0000	0.0000	-0.0000	0.0000	0.0000	0.0000	-0.0000	-0.0000
0.0000	3.2843	0.0000	-1.1533	-0.0000	0.7706	0.0000	-0.6533
-0.0000	0.0000	0.0000	-0.0000	0.0000	0.0000	-0.0000	-0.0000
0.0000	-1.1533	-0.0000	0.4050	0.0000	-0.2706	-0.0000	0.2294
0.0000	-0.0000	-0.0000	0.0000	0.0000	-0.0000	-0.0000	0.0000
0.0000	0.7706	0.0000	-0.2706	-0.0000	0.1808	0.0000	-0.1533
-0.0000	0.0000	-0.0000	-0.0000	-0.0000	0.0000	-0.0000	0.0000
-0.0000	-0.6533	-0.0000	0.2294	0.0000	-0.1533	0.0000	0.1299

Figure 8-Sample inputs and their respective DCT transforms

An interesting behavior about the transform is that the coefficient for the DC basis frequency component would hold the total value of the pixels in the input divided by N, where N is the size of the matrix (NxN). This can be used to gauge how filled a DSM is ranging from 0 to 8 (for an 8x8 matrix); For instance, if the input matrix is fully filled with ones, the DC value would be 8, while if the matrix were only half filled with ones, this DC value would be 4.

Another behavior of interest that can be seen from the above examples is that the more scattered the ones are in the input matrix (for a given number of filled cells), the higher the coefficients of the higher frequency basis frequency components. This is because while you will only need lower frequency basis frequency components to recreate large grouped up blocks of ones (as there are fewer transitions between ones and zeros), a scattered input would require more basis frequency components of higher frequencies to recreate the ones and zeros which are alternating throughout the DSM (as there are more transitions between ones and zeros). In a later section, we will be exploiting this behavior to determine how 'sorted' or 'grouped up' the elements in a DSM are.

Uses of DCT

At the moment, the DCT is currently employed for image compression in image and video files. In this form of lossy compression, an image is first divided in groups of 8x8 pixels and the DCT is applied to these small 8x8 groups of pixels.

The resulting frequency components are then divided element-wise by a quantization matrix which divides the higher frequency components more than the low frequency components. The result is rounded, causing most of the higher frequencies to be rounded down to 0 and unable to be restored when decompressing the image (hence the lossy nature of this compression). As most of the higher frequency components are now 0, using a Huffman code would reduce the number of bits to represent the image, resulting in smaller file sizes. The reverse process is performed for image decompression.

7. Applications of DCT to a DSM

Applying the frequency transform to sorting

The feature of the DCT that we would like to exploit is the non-correlation between adjacent cells in the frequency domain as this would allow us to compare similar looking patterns or behaviors in a time domain DSM by finding the closest match in terms of its frequency transform. The following section explains the process to come up with this scoring system.

First, we will create a scoring matrix in the frequency domain that would characterize the time-domain pattern that we are looking for.

In every DSM, there is a diagonal across the DSM which maps entities onto itself, which is the first transform we will add to our scoring matrix.

Input							
1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1

Transform							
1.0000	0.0000	-0.0000	0.0000	0.0000	0.0000	-0.0000	-0.0000
0.0000	1.0000	0.0000	-0.0000	0.0000	-0.0000	0.0000	0.0000
-0.0000	0.0000	1.0000	0.0000	-0.0000	0.0000	0.0000	0.0000
0.0000	-0.0000	0.0000	1.0000	0.0000	0.0000	-0.0000	0.0000
0.0000	0.0000	-0.0000	0.0000	1.0000	0.0000	-0.0000	-0.0000
0.0000	-0.0000	0.0000	0.0000	0.0000	1.0000	0.0000	-0.0000
-0.0000	0.0000	0.0000	-0.0000	-0.0000	0.0000	1.0000	0.0000
-0.0000	0.0000	0.0000	0.0000	-0.0000	-0.0000	0.0000	1.0000

Figure 9- Input of diagonal line in an 8x8 matrix and its DCT transform

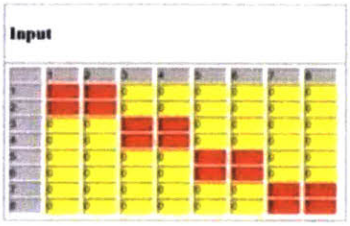
Now we can use the transform above as our scoring matrix if we want our DSM to look like a diagonal line (similar to the above input). In order to do this, we can multiply a transform of any test DSM with the above transform to obtain a score. If we have the exact same transform, the positives would multiply with the positives and the negatives would multiply with the negatives giving us the highest possible score for a given number of relationships. If the DSM we are scoring is different from the transform, the coefficients would not line up with the test matrix's transform and the score would be lower than the highest possible score. This is possible due to the nature of the transform in that an increase in value in one cell would result in a decrease in value in other cells given a specific number of shaded cells in the time domain.

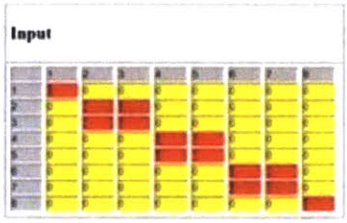
If a DSM looks similar to the input but not exactly the same, the transform would also look similar with some slight variations (especially to the higher frequency coefficients). Following

this behavior, the more similar the DSM looks compared to the scoring matrix input, the higher the score would be when multiplying its transform with that of the scoring matrix.

In order to exploit this for use in DSM sorting to create modules, we need to add a range of these 'scoring matrices' which would let us score clustered DSMs higher than non-clustered ones.

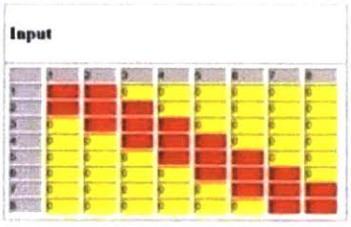
One behavior that we notice is that when forming compact modules, these modules will tend to aggregate around the diagonal to form 'boxes', the next intuitive step would be to add the frequency transform of these 'boxes' to our scoring matrix. These boxes however are arbitrarily positioned along the center diagonal, so we will need to consider the various positions by adding the transforms below:

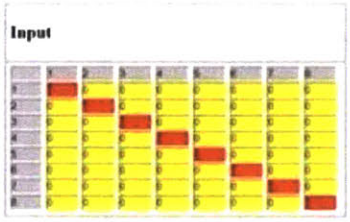
Transform of 

+ Transform of 

--(1)

Adding up the two above transforms would also be equivalent to adding up a thicker diagonal line with redundancies for the transform of the center diagonal itself (depending on how many transforms are added up):

Transform of 

+ 2x Transform of 

--(2)

As the redundancies in the center diagonal are not required for the scoring matrix (it already has been added earlier), we can just add the transform of the thicker diagonal from (2) to the scoring matrix instead of the 2 transforms in (1).

Increasing the size of the clusters would increase the thickness of the line. So, in effect, creating a scoring matrix by adding up the transforms of diagonals with increasing thickness caters for all possible cluster sizes; we obtain a diagonal with varying degrees of thickness, until the whole DSM is filled (in this case, the whole 8x8 DSM will be a module).

As a result, we also add the transforms of the center diagonals with varying thicknesses to the scoring matrix. By adding all these transforms together, a DSM with relationships that cluster around the center diagonal would obtain a higher score compared to a DSM with relationships that do not cluster around the center diagonal.

Input

1	1	0	0	0	0	0	0
1	1	1	0	0	0	0	0
0	1	1	1	0	0	0	0
0	0	1	1	1	0	0	0
0	0	0	1	1	1	0	0
0	0	0	0	1	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	0	0	1	1

Transform

2.7500	0.0000	-0.3266	0.0000	-0.2500	0.0000	-0.1353	-0.0000
-0.0000	2.3668	-0.0000	-0.4077	-0.0000	-0.2724	-0.0000	-0.0957
-0.3266	-0.0000	1.9874	0.0000	-0.3266	-0.0000	-0.1768	-0.0000
0.0000	-0.4077	0.0000	1.4197	0.0000	-0.2310	0.0000	-0.0811
-0.2500	-0.0000	-0.3266	0.0000	0.7500	-0.0000	-0.1353	-0.0000
0.0000	-0.2724	0.0000	-0.2310	-0.0000	0.0803	-0.0000	-0.0542
-0.1353	-0.0000	-0.1768	0.0000	-0.1353	-0.0000	-0.4874	-0.0000
-0.0000	-0.0957	-0.0000	-0.0811	-0.0000	-0.0542	0.0000	-0.8668

Input

1	1	1	0	0	0	0	0
1	1	1	1	0	0	0	0
1	1	1	1	1	0	0	0
0	1	1	1	1	1	0	0
0	0	1	1	1	1	1	0
0	0	0	1	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	0	1	1	1

Transform

4.2500	0.0000	-0.7886	0.0000	-0.2500	0.0000	0.0560	-0.0000
0.0000	2.9655	0.0000	-0.6577	-0.0000	-0.0224	-0.0000	0.0957
-0.7886	0.0000	1.6339	0.0000	-0.1353	-0.0000	0.1768	0.0000
0.0000	-0.6577	0.0000	0.1677	0.0000	0.2310	0.0000	0.1689
-0.2500	-0.0000	-0.1353	0.0000	-0.7500	-0.0000	0.3266	0.0000
0.0000	-0.0224	-0.0000	0.2310	-0.0000	-0.7890	-0.0000	0.1958
0.0560	-0.0000	0.1768	0.0000	0.3266	-0.0000	-0.1339	-0.0000
-0.0000	0.0957	0.0000	0.1689	0.0000	0.1958	-0.0000	0.6558

Input

1	1	1	1	0	0	0	0
1	1	1	1	1	0	0	0
1	1	1	1	1	1	0	0
1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1
0	0	0	1	1	1	1	1
0	0	0	0	1	1	1	1

Transform

5.5000	0.0000	-1.1152	0.0000	0.0000	0.0000	-0.0793	-0.0000
0.0000	2.8403	0.0000	-0.3266	-0.0000	0.1353	0.0000	-0.1353
-1.1152	0.0000	0.5000	0.0000	0.4619	-0.0000	0.0000	-0.0000
0.0000	-0.3266	0.0000	-0.8836	-0.0000	0.3266	0.0000	-0.1353
0.0000	-0.0000	0.4619	-0.0000	-0.5000	0.0000	-0.1913	-0.0000
0.0000	0.1353	-0.0000	0.3266	0.0000	0.4694	0.0000	-0.3266
-0.0793	0.0000	0.0000	0.0000	-0.1913	0.0000	0.5000	0.0000
-0.0000	-0.1353	0.0000	-0.1353	-0.0000	-0.3266	0.0000	-0.4261

Input

1	1	1	1	1	0	0	0
1	1	1	1	1	1	0	0
1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1
0	0	0	1	1	1	1	1

Transform

6.5000	0.0000	-1.1152	0.0000	0.0000	0.0000	-0.0793	-0.0000
0.0000	2.1870	0.0000	0.3266	0.0000	-0.1353	-0.0000	0.1353
-1.1152	0.0000	-0.5000	-0.0000	0.4619	-0.0000	0.0000	0.0000
0.0000	0.3266	-0.0000	-0.6130	-0.0000	-0.3266	-0.0000	0.1353
0.0000	0.0000	0.4619	-0.0000	0.5000	-0.0000	-0.1913	-0.0000
0.0000	-0.1353	-0.0000	-0.3266	-0.0000	0.1988	0.0000	0.3266
-0.0793	-0.0000	0.0000	-0.0000	-0.1913	-0.0000	-0.5000	0.0000
-0.0000	0.1353	0.0000	0.1353	0.0000	0.3266	0.0000	0.2272

Input

1	1	1	1	1	1	0	0
1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1

Transform

7.2500	0.0000	-0.7886	0.0000	-0.2500	0.0000	0.0560	-0.0000
0.0000	1.2965	0.0000	0.6577	0.0000	0.0224	0.0000	-0.0957
-0.7886	0.0000	-0.7803	-0.0000	-0.1353	-0.0000	0.1768	0.0000
0.0000	0.6577	-0.0000	0.1835	-0.0000	-0.2310	-0.0000	-0.1689
-0.2500	0.0000	-0.1353	-0.0000	0.2500	-0.0000	0.3266	0.0000
0.0000	0.0224	-0.0000	-0.2310	-0.0000	-0.3906	-0.0000	-0.1958
0.0560	0.0000	0.1768	0.0000	0.3266	-0.0000	0.2803	0.0000
-0.0000	-0.0957	0.0000	-0.1689	0.0000	-0.1958	0.0000	-0.0894

Input							
1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
0	1	1	1	1	1	1	1

Transform							
7.7500	0.0000	-0.3266	0.0000	-0.2500	0.0000	-0.1353	-0.0000
0.0000	0.4810	-0.0000	0.4077	0.0000	0.2724	0.0000	0.0957
-0.3266	0.0000	-0.4268	-0.0000	-0.3266	-0.0000	-0.1768	0.0000
0.0000	0.4077	-0.0000	0.3457	0.0000	0.2310	0.0000	0.0811
-0.2500	0.0000	-0.3266	0.0000	-0.2500	-0.0000	-0.1353	0.0000
0.0000	0.2724	-0.0000	0.2310	-0.0000	0.1543	-0.0000	0.0542
-0.1353	0.0000	-0.1768	0.0000	-0.1353	-0.0000	-0.0732	0.0000
-0.0000	0.0957	0.0000	0.0811	0.0000	0.0542	0.0000	0.0190

Input							
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1

Transform							
8.0000	0.0000	-0.0000	0.0000	0.0000	0.0000	-0.0000	-0.0000
0.0000	0.0000	0.0000	-0.0000	0.0000	0.0000	-0.0000	-0.0000
-0.0000	0.0000	0.0000	0.0000	-0.0000	0.0000	0.0000	0.0000
0.0000	0.0000	0.0000	0.0000	-0.0000	-0.0000	0.0000	0.0000
0.0000	-0.0000	0.0000	-0.0000	0.0000	-0.0000	-0.0000	0.0000
0.0000	-0.0000	-0.0000	-0.0000	-0.0000	0.0000	-0.0000	-0.0000
-0.0000	-0.0000	0.0000	0.0000	0.0000	-0.0000	-0.0000	0.0000
-0.0000	0.0000	0.0000	0.0000	0.0000	-0.0000	0.0000	0.0000

Figure 10- Inputs and respective transforms of a diagonal line of increasing thickness, to be used in the scoring matrix

Adding up all the transforms for the above, including the transform of the first diagonal line, a score matrix is obtained as such:

43	0	-4.46	0	-1	0	-0.32	0
0	13.14	0	0	0	0	0	0
-4.46	0	3.41	0	0	0	0	0
0	0	0	1.62	0	0	0	0
-1	0	0	0	1	0	0	0
0	0	0	0	0	0.72	0	0
-0.32	0	0	0	0	0	0.59	0
0	0	0	0	0	0	0	0.52

Figure 11-Proposed scoring matrix used for clustering (without penalizing high frequencies)

However, that is not the only behavior we would like to exploit. As mentioned in the previous section, a DSM with its relationships randomly scattered around the DSM would have more higher frequency components than that of a well sorted DSM; there would be a need for higher frequency basis functions to recreate the ones and zeros which are alternating throughout the DSM. On the other hand, if all the ones are grouped up in a block, followed by a block of zeros, the DSM can be recreated with lower frequency basis functions.

By virtue of this, if we penalize the use of high frequency cosines on the scoring matrix, the DSMs which score higher would be those that have more clustered elements. To do this we first create another matrix which increases as we go from the top left to bottom right.

1	2	3	4	5	6	7	8
2	3	4	5	6	7	8	9
3	4	5	6	7	8	9	10
4	5	6	7	8	9	10	11
5	6	7	8	9	10	11	12
6	7	8	9	10	11	12	13
7	8	9	10	11	12	13	14
8	9	10	11	12	13	14	15

Figure 12- 'Quantization matrix' used to penalize the use of high frequencies

By dividing our scoring matrix element-wise with the above we would penalize the use of high frequency cosines and encourage clustering. This would change the scoring matrix to the following:

43	0	-1.49	0	-0.2	0	-0.05	0
0	4.38	0	0	0	0	0	0
-1.49	0	0.68	0	0	0	0	0
0	0	0	0.23	0	0	0	0
-0.2	0	0	0	0.11	0	0	0
0	0	0	0	0	0.07	0	0
-0.05	0	0	0	0	0	0.05	0
0	0	0	0	0	0	0	0.03

Figure 13-Proposed final scoring matrix used for clustering.

In theory, by combing through all permutations of ordering the elements for any arbitrary 8x8 DSM, the transform of the permutation which most resembles the above scoring matrix would be the most likely to exhibit behaviors of lining up along the diagonal line and have their relationships aggregate into clusters.

To put this into practice, we can use an algorithm as such:

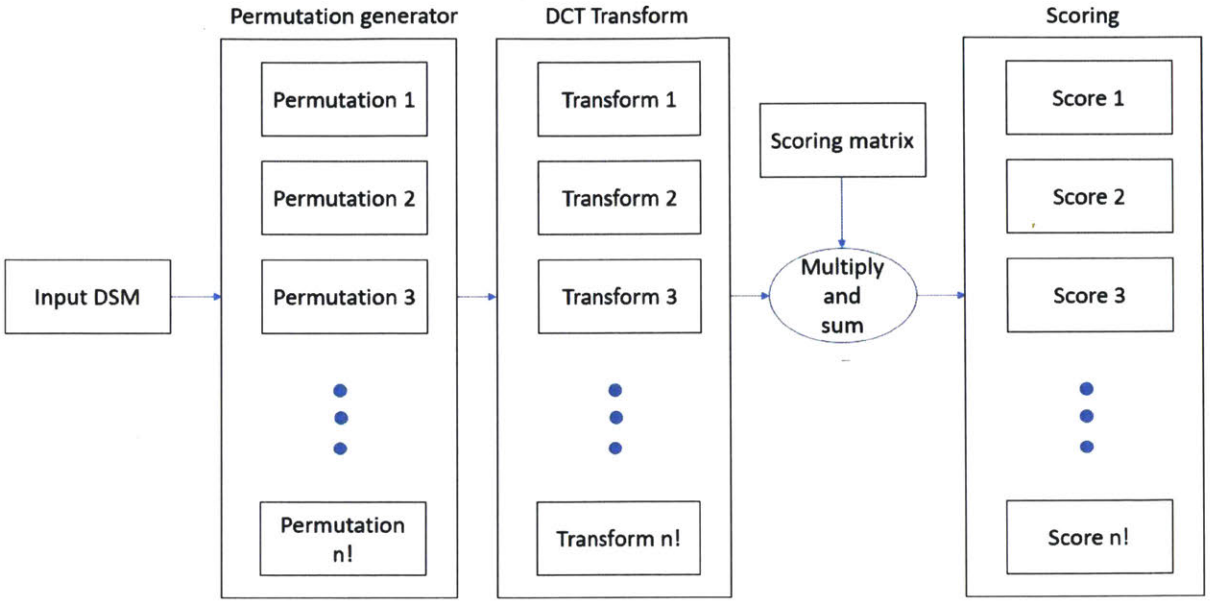


Figure 14- Block diagram showing code functions

The code takes an input DSM and generates all permutations of it by reordering the elements. For each permutation, a DCT transform is performed which is then multiplied elementwise by the scoring matrix and summed to generate a score for each one (a higher score meaning that the DSM is closer to the specified behavior characterized by the scoring matrix). The scores are then sorted in descending order and the top 3 DSMs which scored the highest are displayed. Below are some examples of the input DSMs and their 3 permutations which scored the highest upon running through the clustering algorithm.

Use of algorithm in clustering

Example 1

Example 1 shows the input DSM and the 3 permutations which scored the highest when their transforms are multiplied with the score matrix (high scores are 186.492, 186.492, 186.492 respectively).

Input

	1	2	3	4	5	6	7	8
1	1	0	1	0	1	0	1	0
2	0	1	0	1	0	1	0	1
3	1	0	1	0	1	0	1	0
4	0	1	0	1	0	1	0	1
5	1	0	1	0	1	0	1	0
6	0	1	0	1	0	1	0	1
7	1	0	1	0	1	0	1	0
8	0	1	0	1	0	1	0	1

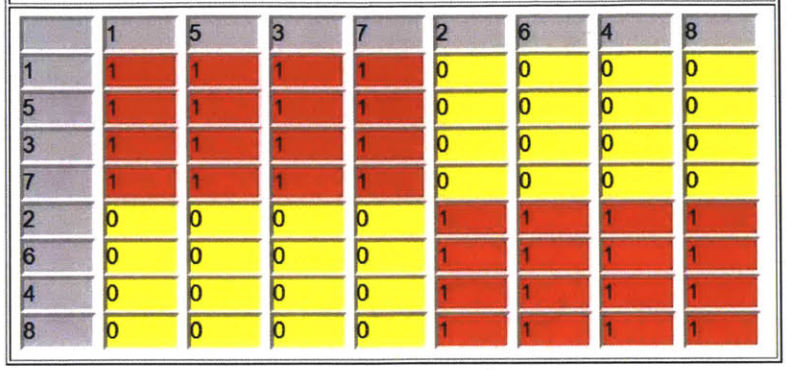
Highscore 1

	5	3	1	7	2	6	4	8
5	1	1	1	1	0	0	0	0
3	1	1	1	1	0	0	0	0
1	1	1	1	1	0	0	0	0
7	1	1	1	1	0	0	0	0
2	0	0	0	0	1	1	1	1
6	0	0	0	0	1	1	1	1
4	0	0	0	0	1	1	1	1
8	0	0	0	0	1	1	1	1

Highscore 2

	3	5	1	7	2	6	4	8
3	1	1	1	1	0	0	0	0
5	1	1	1	1	0	0	0	0
1	1	1	1	1	0	0	0	0
7	1	1	1	1	0	0	0	0
2	0	0	0	0	1	1	1	1
6	0	0	0	0	1	1	1	1
4	0	0	0	0	1	1	1	1
8	0	0	0	0	1	1	1	1

Highscore 3



	1	5	3	7	2	6	4	8
1	1	1	1	1	0	0	0	0
5	1	1	1	1	0	0	0	0
3	1	1	1	1	0	0	0	0
7	1	1	1	1	0	0	0	0
2	0	0	0	0	1	1	1	1
6	0	0	0	0	1	1	1	1
4	0	0	0	0	1	1	1	1
8	0	0	0	0	1	1	1	1

Figure 15- Example 1 input DSM and 3 highest scoring permutations

This input is unlikely to happen in reality but shows an ideal case of the algorithm choosing a rearrangement which is favorable to modularization from the given input.

We run the same input through component partitioning algorithm in Lattix (which is a faster alternative to the Warfield algorithm [10]) to see how the sorting performance matches up to another already existing clustering algorithm. This algorithm, groups strongly connected components into virtual partitions and independent elements without cyclic dependencies together. The elements are then ordered to minimize the weight above the diagonal in the DSM. Once elements are grouped, topological sort is used to order them so that dependencies between the groupings are in the same direction.

Do note that in order to make a comparison, we cannot compare the exact location of every dependency as there will be for any given output many ways to represent the DSM. To illustrate this, if I were to have a system with 3 different groups of modules, where I place these groups relative to one another along the center diagonal would change the ordering of the DSM, but the solution is essentially the same. An important point for comparison would be the performance of the algorithm on its intent. In the case of clustering algorithms, the intent is to modularize systems to reduce interfaces between modules. In order to compare this performance, if the same number and size of modules are overlaid on the outputs of 2 different algorithms (in any position along the center diagonal) and the number of external dependencies found outside these modules are the same, the 2 algorithms should be considered equally successful in their intent to reduce interfaces. If there is a larger number of external dependencies found in a particular output, the algorithm which arranged it that way could be said to be less successful in its intent to reduce interfaces between modules.

The result is shown below that both algorithms exhibit equivalent performance for this example:

	1	2	3	4	5	6	7	8
1	1		1		1		1	
2		1		1		1		1
3	1		1		1		1	
4		1		1		1		1
5	1		1		1		1	
6		1		1		1		1
7	1		1		1		1	
8		1		1		1		1

Figure 16-Showing input into Lattix component partitioning algorithm (Example 1)

	2	4	6	8	1	3	5	7
2	1	1	1	1				
4	1	1	1	1				
6	1	1	1	1				
8	1	1	1	1				
1					1	1	1	1
3					1	1	1	1
5					1	1	1	1
7					1	1	1	1

Figure 17- Result after sorting with Lattix component partitioning algorithm (Example 1)

Example 2

Example 2 shows the input DSM and the 3 permutations which scored the highest when their transforms are multiplied with the score matrix (high scores are 127.128, 127.128, 127.112 respectively). We see that the dependencies on the outer edges of the input DSM are aggregated around the center diagonal for the higher scoring permutations.

Input

	1	2	3	4	5	6	7	8
1	1	1	1	0	1	0	0	1
2	0	1	0	0	0	1	0	0
3	0	0	1	0	0	0	0	0
4	0	0	0	1	0	0	1	1
5	1	0	0	0	1	0	0	0
6	1	0	0	0	0	1	0	1
7	0	1	1	0	0	0	1	1
8	0	0	0	0	1	0	0	1

Highscore 1

	6	2	1	5	8	7	3	4
6	1	0	1	0	1	0	0	0
2	1	1	0	0	0	0	0	0
1	0	1	1	1	1	0	1	0
5	0	0	1	1	0	0	0	0
8	0	0	0	1	1	0	0	0
7	0	1	0	0	1	1	1	0
3	0	0	0	0	0	0	1	0
4	0	0	0	0	1	1	0	1

Highscore 2

	4	3	7	8	5	1	2	6
4	1	0	1	1	0	0	0	0
3	0	1	0	0	0	0	0	0
7	0	1	1	1	0	0	1	0
8	0	0	0	1	1	0	0	0
5	0	0	0	0	1	1	0	0
1	0	1	0	1	1	1	1	0
2	0	0	0	0	0	0	1	1
6	0	0	0	1	0	1	0	1

Highscore 3

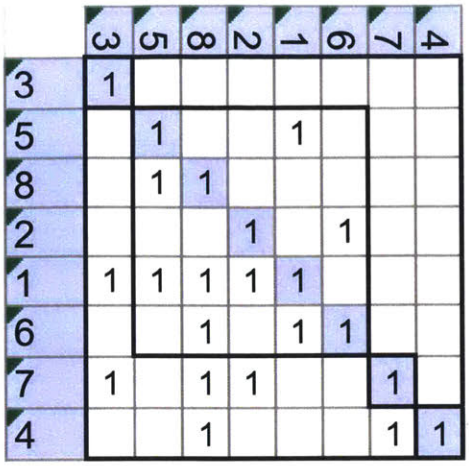
	4	3	7	8	5	1	6	2
4	1	0	1	1	0	0	0	0
3	0	1	0	0	0	0	0	0
7	0	1	1	1	0	0	0	1
8	0	0	0	1	1	0	0	0
5	0	0	0	0	1	1	0	0
1	0	1	0	1	1	1	0	1
6	0	0	0	1	0	1	1	0
2	0	0	0	0	0	0	1	1

Figure 18-Example 2 input DSM and 3 highest scoring permutations

When we compare the result of the Warfield[10] algorithm, we obtain the same performance when considering equivalent module sizes (one 5-element module and three 1-element modules). In both cases, the number of dependencies that can be captured within the modules by both algorithms are the same. Conversely, there are 6 dependencies that are not captured within the modules by both the Warfield[10] algorithm and the DCT scoring algorithm when the modules are overlaid on the DSMs.

	1	2	3	4	5	6	7	8
1	1	1	1		1			1
2		1				1		
3			1					
4				1			1	1
5	1				1			
6	1					1		1
7		1	1				1	1
8					1			1

Figure 19-Showing input into Lattix component partitioning algorithm (Example 2)



	3	5	8	2	1	6	7	4
3	1							
5		1			1			
8		1	1					
2				1		1		
1	1	1	1	1	1			
6			1		1	1		
7	1		1	1			1	
4			1				1	1

Figure 20- Result after sorting with Lattix component partitioning algorithm (Example 2)

Example 3

Example 3 shows the input DSM and the 3 permutations which scored the highest when their transforms are multiplied with the score matrix (high scores are 145.982, 145.982, 145.970 respectively). Again, note that the high scores generally exhibit clustering along the center diagonal. This configuration makes it favorable for the system architect to identify clusters around the center diagonal to constitute the modules in a system.

Input

	1	2	3	4	5	6	7	8
1	1	0	1	1	1	0	0	1
2	0	1	0	0	0	0	0	0
3	1	0	1	1	1	0	1	0
4	0	0	0	1	0	0	0	0
5	0	0	1	0	1	0	1	1
6	1	1	0	1	0	1	0	0
7	0	0	1	0	0	0	1	0
8	0	0	1	0	1	0	0	1

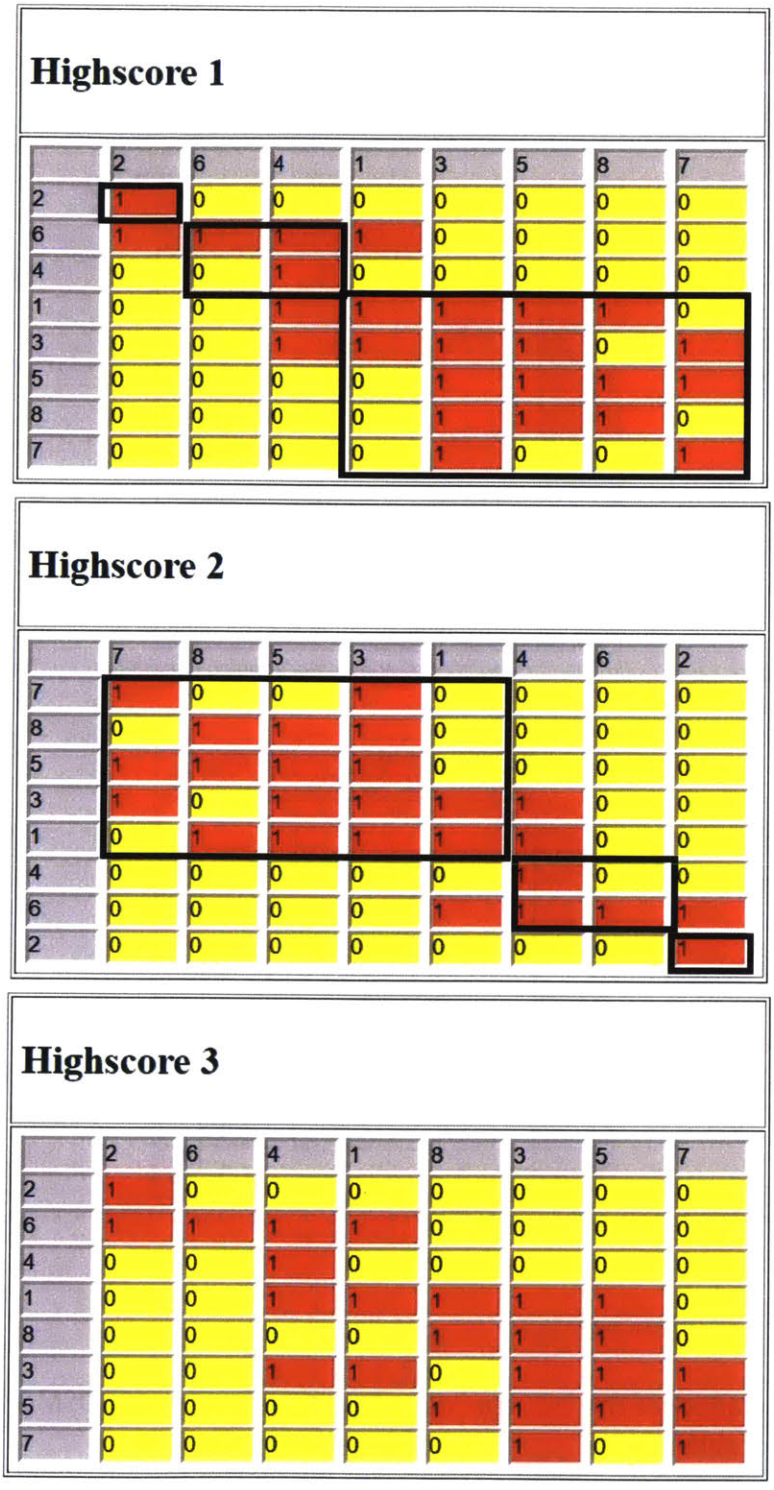


Figure 21-Example 3 input DSM and 3 highest scoring permutations

Once again, when we compare the result of the Warfield[10] algorithm, we obtain a similar performance when considering equivalent module sizes (one 5-element module, one 2-element module and one 1-element module). In both cases, the number of dependencies that can be

captured within the modules by both clustering algorithms are the same. Conversely, there are 4 dependencies that are not captured within the modules by both the Warfield[10] algorithm and the DCT scoring algorithm when the modules are overlaid on the DSMs.

	1	2	3	4	5	6	7	8
1	1		1	1	1			1
2		1						
3	1		1	1	1		1	
4				1				
5			1		1		1	1
6	1	1		1		1		
7			1				1	
8			1		1			1

Figure 22-Showing input into Lattix component partitioning algorithm (Example 3)

	4	2	7	3	8	5	1	6
4	1							
2		1						
7			1	1				
3	1		1	1		1	1	
8				1	1	1		
5			1	1	1	1		
1	1			1	1	1	1	
6	1	1					1	1

Figure 23-Result after sorting with Lattix component partitioning algorithm (Example 3)

Use of algorithm in sequencing

To illustrate the versatility of the DCT transform's sorting potential in DSMs, let us try applying it to another DSM problem.

For a time-based DSM, unlike a static DSM where elements exist simultaneously, there is a timeline as to when activities occur. The ordering of these activities is typically structured such that they are performed when all of their inputs are available (which may be an output of another activity). If the input to an activity is not ready when it is performed, it has to make do with assumptions in place of the actual input and this may lead to downstream rework if the

assumptions are incorrect. On a DSM, if the activities are represented as the elements ordered in time from left to right, dependencies above the center diagonal constitute input requirements which are not ready at the time of the respective activity as the activity leading to the creation of that input has not been completed yet. On the other hand, dependencies below the center diagonal denote inputs to the respective activity whose prerequisite activity has already been completed[8].

Rearranging the DSM such that the relationships are all below the center diagonal will reduce 'feedback' from later activities into earlier activities, resulting in less possible iteration due to this 'feedback'. This is also known as 'sequencing' and is the most common method of analysis applied to process architecture[8].

For this problem, in a process architecture DSM, we would like to minimize the iterations that a process DSM has by arranging the activities with as many dependencies as possible below the center diagonal, gathering as many elements on the bottom half of the DSM. To achieve this, the DSM whose transform we need to use as our scoring matrix would look like the block triangular matrix as described by Warfield[10] in his algorithm, with all the zeros found to the right of the center diagonal and the ones to the left:

1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0
1	1	1	0	0	0	0	0
1	1	1	1	0	0	0	0
1	1	1	1	1	0	0	0
1	1	1	1	1	1	0	0
1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1

Figure 24- DSM of scoring matrix for sequencing (before DCT transform)

Performing a 2-dimensional DCT on the above this would yield the matrix below, which we will use as our scoring matrix. One interesting feature, this scoring matrix is not symmetrical about the center diagonal unlike the one from the previous problem. This is because the DSM for which this transform is derived is not symmetrical about the center diagonal as well. There is a transition from ones to zeros in the input DSM when going from left to right and a transition from zeros to ones when going from top to bottom. This translates into a positive sign when moving from left to right and a negative sign when moving from top to bottom of the transform matrix.

4.5	2.28	0	0.24	0	0.07	0	0.02
-2.28	0.5	1.05	0	0.19	0	0.06	0
0	-1.05	0.5	0.59	0	0.12	0	0.03
-0.24	0	-0.59	0.5	0.38	0	0.07	0
0	-0.19	0	-0.38	0.5	0.26	0	0.04
-0.07	0	-0.12	0	-0.26	0.5	0.16	0
0	-0.06	0	-0.07	0	-0.16	0.5	0.09
-0.02	0	-0.03	0	-0.04	0	-0.09	0.5

Figure 25-DSM of scoring matrix for sequencing (after DCT transform)

Once again, we run through all permutations of a DSM and multiply each of their transforms with this scoring matrix to obtain a score for each one. The scores are then sorted in descending order and the top 3 DSMs which scored the highest are displayed. Below are some examples of the input DSMs and their 3 permutations which scored the highest upon running through the program using this particular scoring matrix.

Example 4

Example 4 shows the input DSM and the 3 permutations which scored the highest when their transforms are multiplied with the score matrix (high scores are 36.00, 35.00, 35.00 respectively). This example acts as a control to show that the highest scoring permutation would have its relationships all under the center diagonal if the input is also a DSM with its relationships already under the center diagonal.

Input

	1	2	3	4	5	6	7	8
1	1	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0
3	1	1	1	0	0	0	0	0
4	1	1	1	1	0	0	0	0
5	1	1	1	1	1	0	0	0
6	1	1	1	1	1	1	0	0
7	1	1	1	1	1	1	1	0
8	1	1	1	1	1	1	1	1

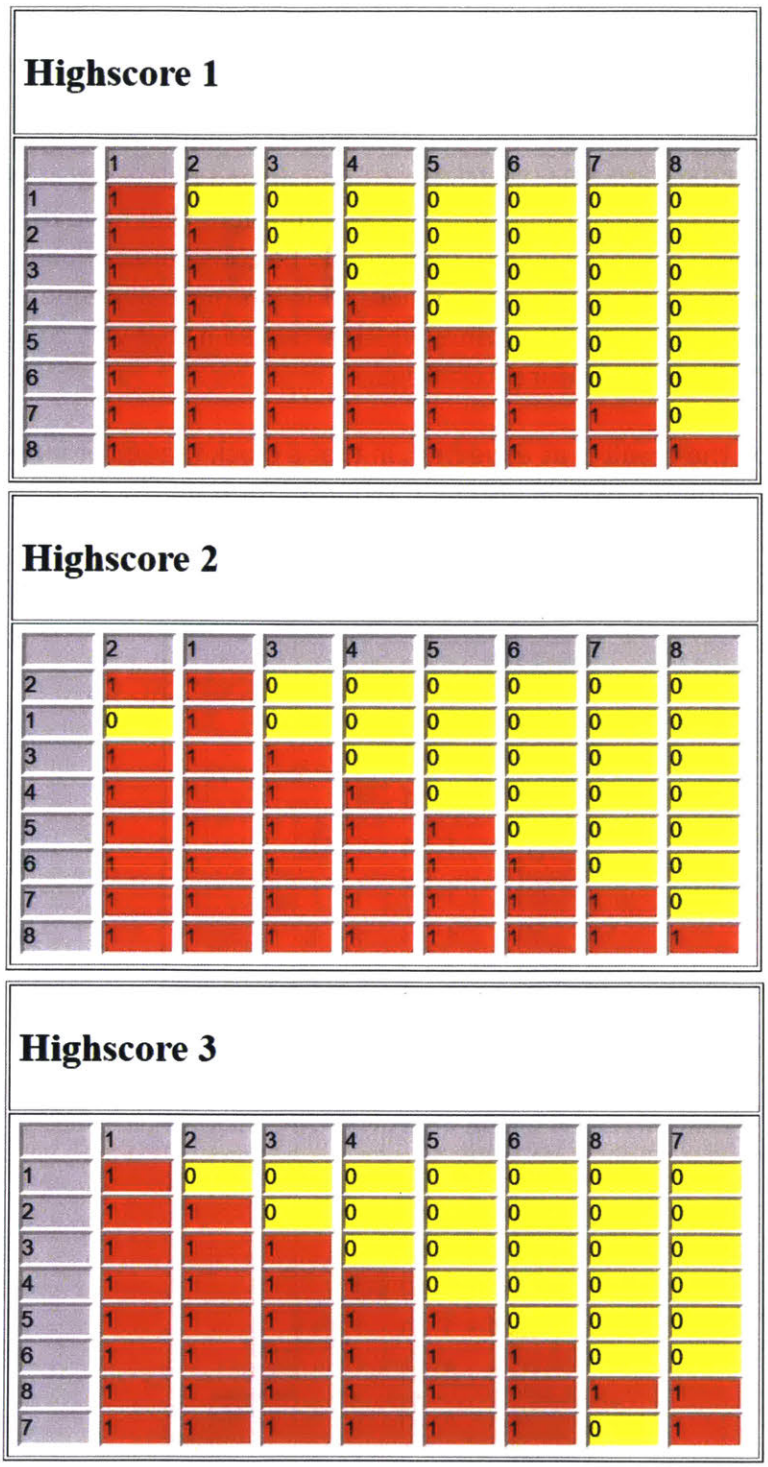


Figure 26- Example 4 input DSM and its 3 highest scoring permutations

We can compare the performance of the result with an existing sequencing algorithm used in Lattix 'As Early As Possible' algorithm based off the sequencing algorithm by S.Cho[16].

Again, in order to gauge the performance of both algorithms, we cannot look at the absolute positions of each dependency, but rather at how well each algorithm fulfils its intent. Unlike clustering, the intent for sequencing is to reduce iterations by moving dependencies below the center diagonal. In order to compare the performance of a sequencing algorithm, we should look at the number of dependencies which have been moved below the center diagonal line. If the output of two different algorithms result in configurations that have the same number of dependencies below the center diagonal, they should be considered equally successful in their intent to reduce iterations. If a particular configuration has more dependencies below the center diagonal than another, the algorithm which arranged it that way could be said to be more successful in its intent to reduce iterations.

As this is the control, the result is as expected, in that a block triangular input matrix which is already the optimum solution would also be the output for both the algorithms being compared.

	1	2	3	4	5	6	7	8
1	1							
2	1	1						
3	1	1	1					
4	1	1	1	1				
5	1	1	1	1	1			
6	1	1	1	1	1	1		
7	1	1	1	1	1	1	1	
8	1	1	1	1	1	1	1	1

Figure 27-Showing input into Lattix 'As Early As Possible' algorithm (Example 4)

	1	2	3	4	5	6	7	8
1	1							
2	1	1						
3	1	1	1					
4	1	1	1	1				
5	1	1	1	1	1			
6	1	1	1	1	1	1		
7	1	1	1	1	1	1	1	
8	1	1	1	1	1	1	1	1

Figure 28-Result after sorting with Lattix 'As Early As Possible' algorithm (Example 4)

Example 5

Example 5 shows the input DSM and the 3 permutations which scored the highest when their transforms are multiplied with the score matrix (high scores are 21.000099, 21.000095, 21.000089 respectively). Note that there are no possible permutations with all the relationships under the diagonal line given the input, which is why the highest scoring permutation still has one relationship above the center diagonal.

Input

	1	2	3	4	5	6	7	8
1	1	0	0	0	0	0	0	1
2	0	1	0	0	1	0	0	0
3	0	0	1	0	0	0	1	0
4	1	1	0	1	0	1	0	1
5	0	0	0	0	1	0	1	0
6	1	0	1	0	0	1	0	1
7	0	0	0	0	0	0	1	0
8	1	0	1	0	1	0	0	1

Highscore 1

	7	1	5	3	2	8	6	4
7	1	0	0	0	0	0	0	0
1	0	1	0	0	0	1	0	0
5	1	0	1	0	0	0	0	0
3	1	0	0	1	0	0	0	0
2	0	0	1	0	1	0	0	0
8	0	1	1	1	0	1	0	0
6	0	1	0	1	0	1	1	0
4	0	1	0	0	1	1	1	1

Highscore 2

	7	1	5	3	8	2	6	4
7	1	0	0	0	0	0	0	0
1	0	1	0	0	1	0	0	0
5	1	0	1	0	0	0	0	0
3	1	0	0	1	0	0	0	0
8	0	1	1	1	1	0	0	0
2	0	0	1	0	0	1	0	0
6	0	1	0	1	1	0	1	0
4	0	1	0	0	1	1	1	1

Highscore 3

	7	1	3	5	2	8	6	4
7	1	0	0	0	0	0	0	0
1	0	1	0	0	0	1	0	0
3	1	0	1	0	0	0	0	0
5	1	0	0	1	0	0	0	0
2	0	0	0	1	1	0	0	0
8	0	1	1	1	0	1	0	0
6	0	1	1	0	0	1	1	0
4	0	1	0	0	1	1	1	1

Figure 29- Example 5 input DSM and 3 highest scoring permutations

Comparing it with the S.Cho[16] algorithm yields a result with equivalent performance, in that after sorting, only one dependency is left above the center diagonal line in both cases; seven opportunities for rework was reduced to one opportunity for rework after sequencing in both algorithms.

	1	2	3	4	5	6	7	8
1	1							1
2		1			1			
3			1				1	
4	1	1		1		1		1
5					1		1	
6	1		1			1		1
7							1	
8	1		1		1			1

Figure 30-Showing input into Lattix 'As Early As Possible' algorithm (Example 5)

	7	5	3	8	1	2	6	4
7	1							
5	1	1						
3	1		1					
8		1	1	1	1			
1				1	1			
2		1				1		
6			1	1	1		1	
4				1	1	1	1	1

Figure 31-Result after sorting with Lattix 'As Early As Possible' algorithm (Example 5)

Example 6

Example 6 shows the input DSM and the 3 permutations which scored the highest when their transforms are multiplied with the score matrix (high scores are 27.000075, 27.000062, 27.000062 respectively). Again, there are no possible permutations with all the relationships under the diagonal line given the input, which is why the highest scoring permutations still have relationships above the center diagonal. In essence, the high scores would contain configurations which are the algorithm's best effort to sequence the elements such that they all fall under the center diagonal.

Input

	1	2	3	4	5	6	7	8
1	1	0	0	0	0	0	1	0
2	0	1	0	0	0	1	0	0
3	1	1	1	1	0	1	1	1
4	1	0	0	1	0	0	0	0
5	0	0	1	1	1	0	1	1
6	0	1	0	1	0	1	1	0
7	1	1	0	0	0	1	1	1
8	0	1	0	1	1	0	0	1

Highscore 1

	1	2	4	8	6	7	3	5
1	1	0	0	0	0	1	0	0
2	0	1	0	0	1	0	0	0
4	1	0	1	0	0	0	0	0
8	0	1	1	1	0	0	0	1
6	0	1	1	0	1	1	0	0
7	1	1	0	1	1	1	0	0
3	1	1	1	1	1	1	1	0
5	0	0	1	1	0	1	1	1

Highscore 2

	1	4	2	8	6	7	3	5
1	1	0	0	0	0	1	0	0
4	1	1	0	0	0	0	0	0
2	0	0	1	0	1	0	0	0
8	0	1	1	1	0	0	0	1
6	0	1	1	0	1	1	0	0
7	1	0	1	1	1	1	0	0
3	1	1	1	1	1	1	1	0
5	0	1	0	1	0	1	1	1

Highscore 3

	1	4	2	6	8	7	3	5
1	1	0	0	0	0	1	0	0
4	1	1	0	0	0	0	0	0
2	0	0	1	1	0	0	0	0
6	0	1	1	1	0	1	0	0
8	0	1	1	0	1	0	0	1
7	1	0	1	1	1	1	0	0
3	1	1	1	1	1	1	1	0
5	0	1	0	0	1	1	1	1

Figure 32- Example 6 input DSM and 3 highest scoring permutations

Comparing it with the S.Cho[16] algorithm shows slightly better performance when using the DCT scoring algorithm; nine opportunities for rework was reduced to four opportunities when

using the DCT scoring algorithm as opposed to nine opportunities for rework being reduced to five opportunities when using the 'As Early as Possible' algorithm.

	1	2	3	4	5	6	7	8
1	1						1	
2		1				1		
3	1	1	1	1		1	1	1
4	1			1				
5			1	1	1		1	1
6		1		1		1	1	
7	1	1				1	1	1
8		1		1	1			1

Figure 33-Showing input into Lattix 'As Early As Possible' algorithm (Example 6)

	1	4	2	6	7	8	3	5
1	1				1			
4	1	1						
2			1	1				
6		1	1	1	1			
7	1		1	1	1	1		
8		1	1			1		1
3	1	1	1	1	1	1	1	
5		1			1	1	1	1

Figure 34-Result after sorting with Lattix 'As Early As Possible' algorithm (Example 6)

Advantages and limitations of using this approach

Using this approach to sort DSMs confers certain advantages. Firstly, the scoring matrix of the algorithm drives the process; changing the scoring matrix can entirely change which permutation would be scored the highest and creates versatility in what the algorithm can do. In essence, this algorithm offers a top down rather than a bottoms-up approach to sorting DSMs which is conducive to process improvement.

This versatility also allows this algorithm to sort DSMs differently depending on the scoring matrix used for a wide variety of applications (for example, one scoring matrix can be used for clustering, while another is used for sequencing, as per the earlier examples).

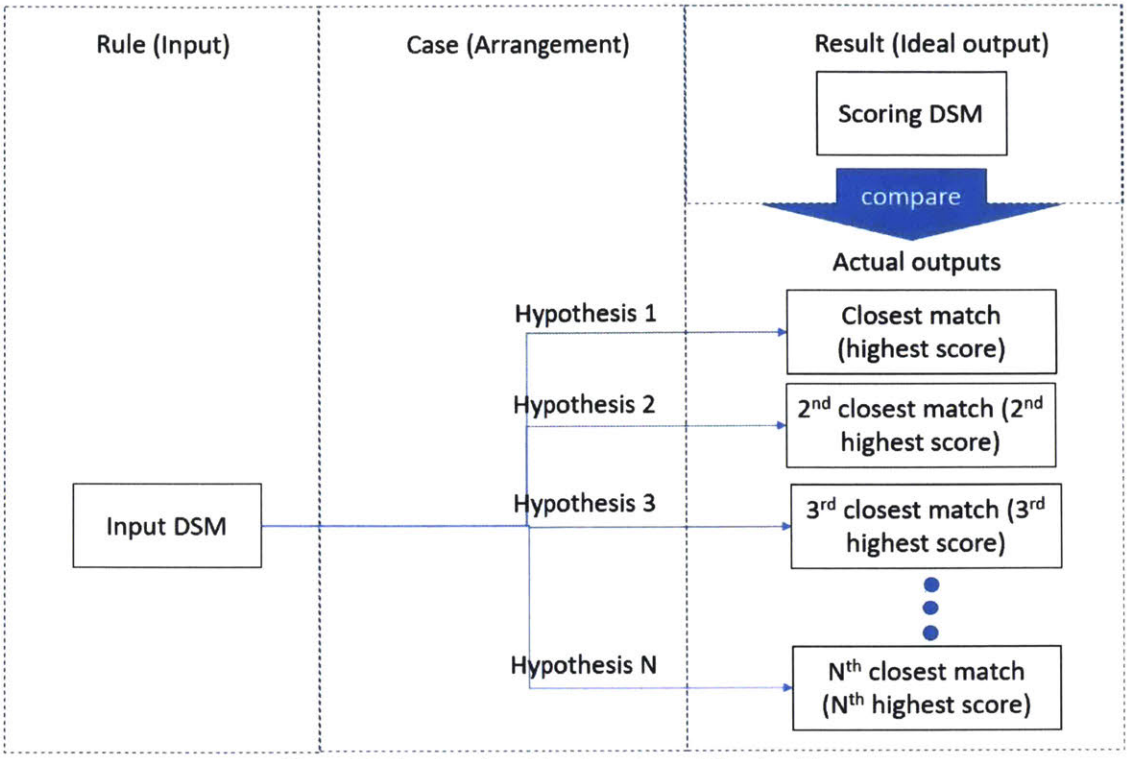


Figure 35-Showing the abductive nature of the algorithm

Tying this back to the abductive approach discussed earlier, the 'rule', which is the input DSM and the 'result', which is the scoring matrix, are initially defined. Rearranging the input DSM gives rise to various hypotheses, constituting the 'case', whose transforms when compared to the scoring matrix, allows us to rank these hypotheses to find out which conform more to the ideal output and are hence more useful to us (high scores). This approach also gives a 'best effort' effect to the high-scoring permutations. If no permutations exist whose transform directly matches the scoring matrix, the next best matrix is chosen which shows up as the highest score.

The main draw of the abductive approach is that this algorithm offers multiple solutions (by looking at a set of highest scores) for the user to make his or her decisions upon rather than a single solution. This may help the him or her develop data points for a tradespace exploration or give him insight on certain useful configurations that he may have overlooked.

Performance-wise in terms of the end result, this algorithm is on par with other existing clustering algorithms. When compared with the Warfield[10] algorithm, this algorithm was able to rank the various DSM permutations such that the highest scoring permutations had an equivalent number of dependencies captured within the modules (when overlaying modules of the same size which the Warfield[10] algorithm defined). When compared with the S.Cho[16] algorithm for sequencing, this algorithm was able to match the performance of reducing rework opportunities. In one instance of sequencing, this algorithm showed slightly better performance than the S.Cho[16] algorithm in that one additional 'feedback' loop was eliminated. This is not a

statistically significant sample of cases for a good comparison, which could be an area of future work on this topic.

Due to the computationally intensive nature of the algorithm (n! computations– where n is the number of elements in the DSM) to run through all permutations of a DSM and score each one. There is a longer computational time required for this algorithm compared to the Warfield[10] and S.Cho[16] algorithm (around 20 seconds as compared to 2 seconds to work on an 8 element matrix).

Applying the frequency transform as a metric

It is entirely possible to use the same scoring method to compare 2 DSMs in a pairwise manner to see which one conforms better to a specific behavior in order to determine a dominant choice. This is similar to using a Hidden Markov Model approach to determine the modularity of a module or system.

An application of this would be to use the same frequency based scoring system to determine and find the optimal module cut for a given system. By determining a metric for both intra-module modularity and overall system modularity by scoring the system's module and overall frequency transforms to the clustering behavior discussed earlier.

The optimal module cut problem

How big should the modules in our system be? How many modules should our system have? These questions constitute the problem of what the optimal module cut should be. While some believe that this components should be divided into portions which are easily managed by the average human mind, consisting of 5 to 9 elements per module[20], there are some who use a upper limit of 20 elements per module [11]. As every system is different, there is no optimal number to use which would apply across the board. This section will attempt to address this problem.

We have discussed earlier that using clustering algorithms to modularize a system would make a system 'better' in terms of reducing its interfaces.

Ultimately however, as with any architectural decision, the value of a system's modularity depends on its stakeholders' needs (which would most likely include some degree of modularity for the system to be more economical in terms of its interfaces). Consequently, and by extension, the optimal number of elements in a module would also be driven by stakeholder needs.

But stakeholders do not normally have a need or requirement for number of elements in a module right? Let us delve into the tradeoffs made when increasing the size of a module using a simple system dynamics model:

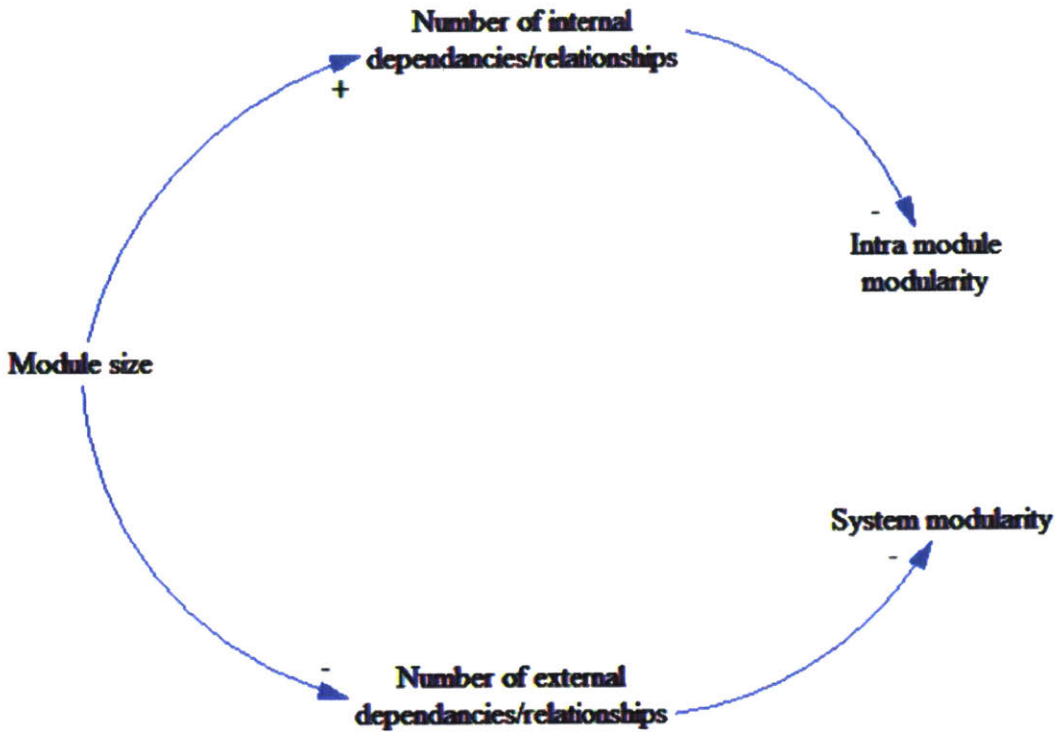


Figure 36- System dynamics diagram showing relationships between module size vs system modularity and intra-module modularity

By increasing module size of a module in a DSM with a fixed number of relationships, there will be more relationships or dependencies captured within said module. If a clustering algorithm were used to cluster the module itself, it would be marginally more difficult with one more dependency to handle. By virtue of this, module size has an inversely proportional relationship to intra-module modularity.

On the other hand, increasing the size of a module would reduce the number of possible relationships or dependencies outside of the module, which would marginally reduce the number of dependencies between modules, improving overall system modularity. This means module size has a directly proportional relationship to overall system modularity.

As having both intra-module modularity and system modularity can be of value to stakeholders, this balance between intra-module modularity and system modularity would be the tradeoff that the stakeholders will need to make when specifying an optimal module cut.

To illustrate, the table below gives two examples of the type of stakeholder needs that would drive a preference to either side of this tradeoff:

Values Intra-module modularity more	Values system modularity more
A stakeholder who wants to future-proof his or her system by making module subcomponents easily interchangeable so that they can be easily changed out when obsolete.	A stakeholder who wants to reduce the costs of interfaces between modules and does not have any plans for a mid-life replacement of obsolete components.

Table 2- Showing the needs of stakeholders and what they value

Both these stakeholder needs are entirely valid, and rather than just being two mutually exclusive, binary states, there is a whole spectrum of where the stakeholder is in terms of his or her preference towards intra-module modularity vs system modularity.

The next section will attempt to use the behavioral/pattern matching feature of the DSM's frequency transform scoring system described earlier in the paper to determine a metric for intra-module modularity and system modularity so that this tradeoff can be visualized in a tradespace.

Using the DCT to determine intra-module modularity and overall system modularity

Below we see a DSM decomposition of a Pratt and Whitney engine (PW4098) consisting of 54 elements as presented in the book 'Design Structure Matrix Methods and Applications' by Eppinger and Browning [8].

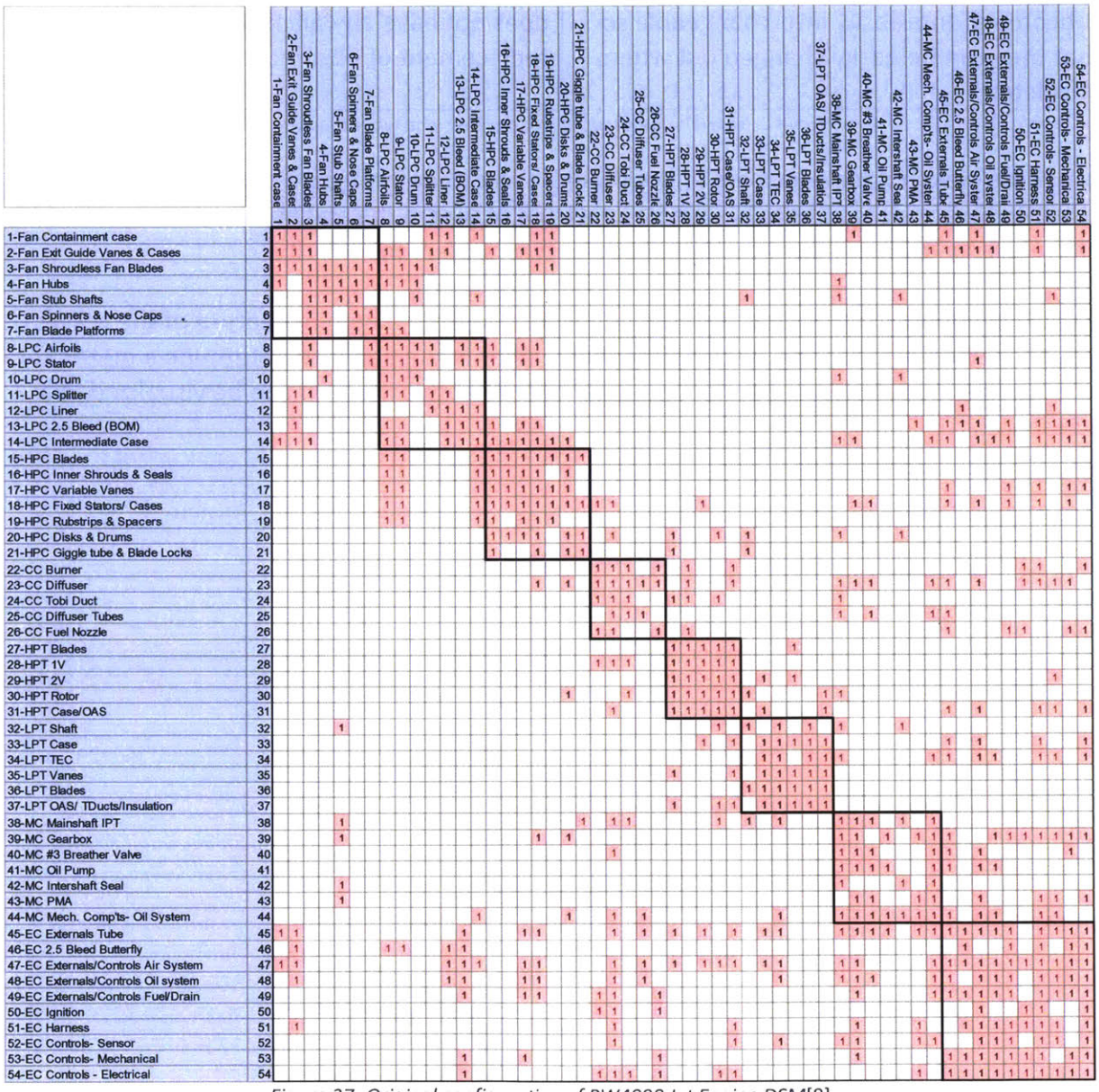


Figure 37- Original configuration of PW4098 Jet Engine DSM[8]

This DSM is clustered into 8 subsystems, 6 of which are somewhat modular with few external dependencies, with 2 integrative or 'bus' subsystems in the bottom right corner. These bus subsystems are highly connected to all the other subsystems and hence have many external dependencies.

This configuration is in itself a perfectly valid solution as to how to modularize the system. In this case, having bus subsystems represented in the DSM acts as a bridge to allow dependencies between other components to be connected to the bus instead. This concentrates the external dependencies on the bus and reduces other non-bus component to component interfaces allowing other components to achieve relatively small sizes with few external dependencies. This is very similar to the Baldwin and Clark 2004[5] algorithm discussed earlier which uses a design rules block instead of a bus. One key architectural decision of having a configuration in this way is that it does not serve to reduce the overall number of external dependencies as they are merely shifted to the 'bus' subsystems. If there was a need to reduce the overall number of external dependencies, other clustering algorithms could be used to rearrange and partition the system without having a 'bus' subsystem, however this would also likely lead to larger module sizes.

Scoring the frequency transform of this DSM configuration yields an overall modular score of 0.000088 and an average intra modular score of 0.051. There are 338 external dependencies outside of the 8 modules in this configuration.

For the purpose of this paper we would be taking the above DSM (referred to as the anchor architecture) and re-clustering the relationships with another arbitrary clustering algorithm into systems with various module sizes. The resulting DSMs would then be analyzed by performing a frequency transform on each module and comparing with a scoring matrix to determine the average internal modularity score for each configuration.

A clustering algorithm called 'Spectral (normalized)' by Hagen and Kahng [21][22] was used to re-cluster it into configurations with a varying number of modules and module sizes. This clustering algorithm breaks up the system into 2 sets, rearranging the elements such that highly dependent elements are put close to each other, and normalizing the number of dependencies in each of the sets. The intent of this algorithm is for it to be run iteratively in order to find natural partitions in the system. By running the algorithm iteratively on the largest module, we obtain different configurations of the PW4098 with an increasing number of modules.

A similar code to what was used previously with the DCT was used to score each of the modules. However, as the size of the modules differ from each other, separate scoring matrixes of different sizes were procedurally generated for each module based on the same principles as the generated 8x8 scoring matrix used earlier. Upon scoring the DCT transform of each module, the scores were then averaged over the size of each module (for example, the score for a 10-element module would have double the weight as compared to a 5-element module) to obtain the overall average intra-modular score for each configuration. The overall modular score is obtained from scoring the entire system as a whole module.

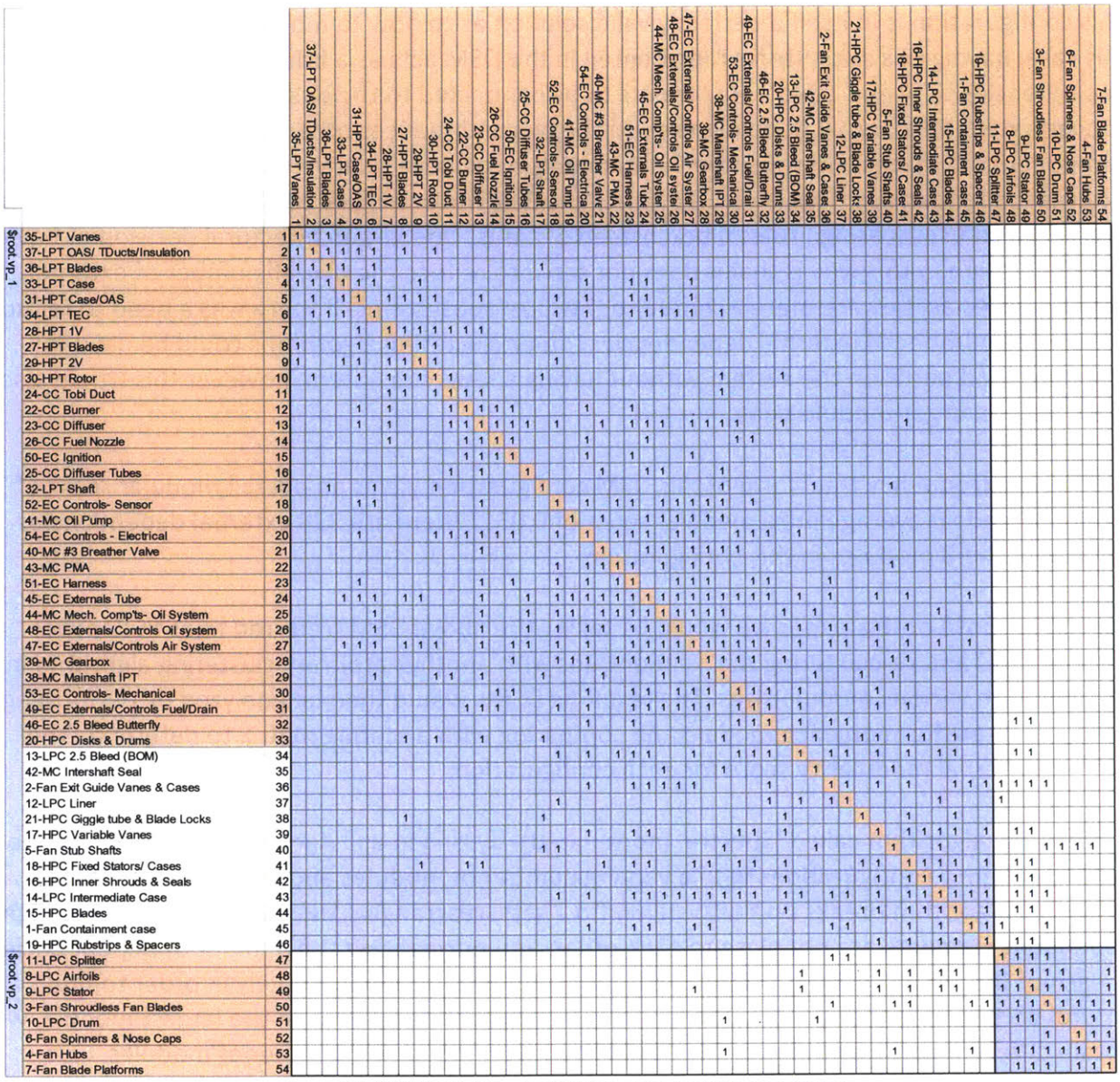


Figure 38- PW4098 Jet Engine DSM re-clustered into 2 modules

Performing a DCT and scoring the frequency transform yields an overall modular score of 8.11E-05 and an average intra modular score of 0.0037 (average taking into account the size of each module). There are 51 external dependencies outside of the 2 modules in this configuration.

Running the algorithm recursively on the larger module yields 3, 4, 5, 6, 7 and 8 module configurations:

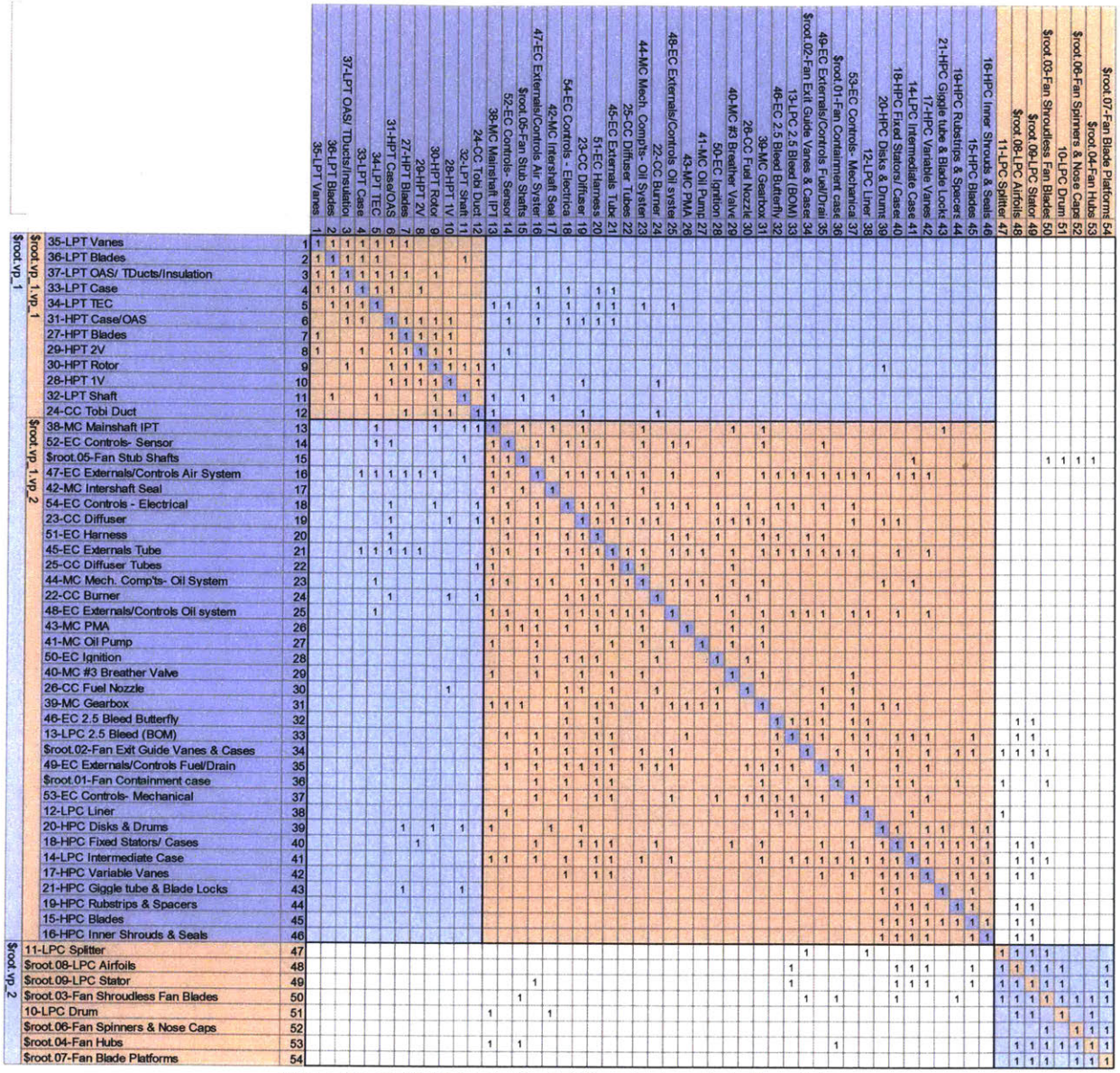


Figure 39- PW4098 Jet Engine DSM re-clustered into 3 modules

Scoring the frequency transform of the 3-module configuration yields an overall modular score of 8.00E-05 and an average intra modular score of 0.0049. There are 118 external dependencies outside of the 3 modules in this configuration.

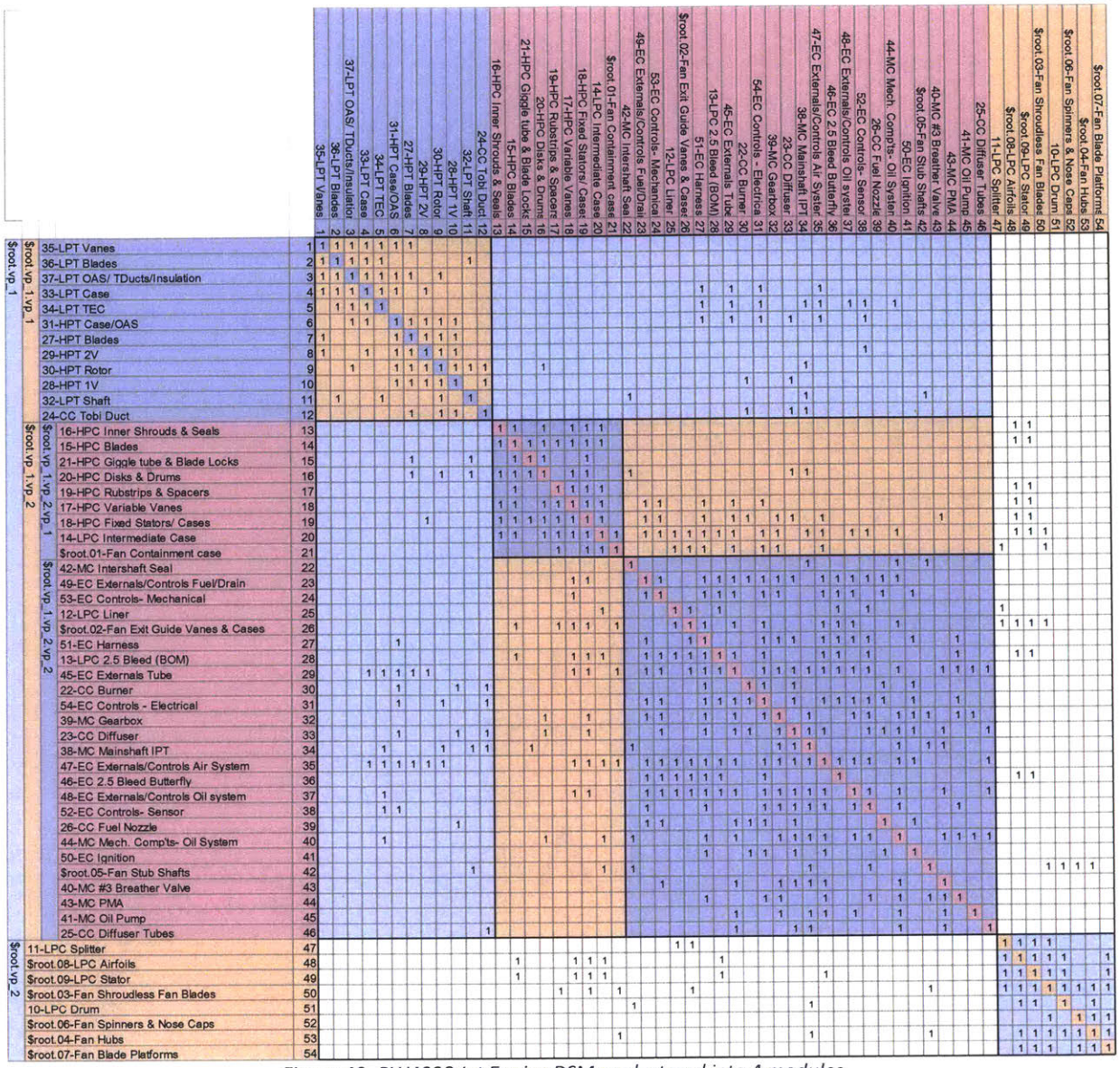


Figure 40- PW4098 Jet Engine DSM re-clustered into 4 modules

Scoring the frequency transform of the 4-module configuration yields an overall modular score of 8.71E-05 and an average intra modular score of 0.0074. There are 186 external dependencies outside of the 4 modules in this configuration.

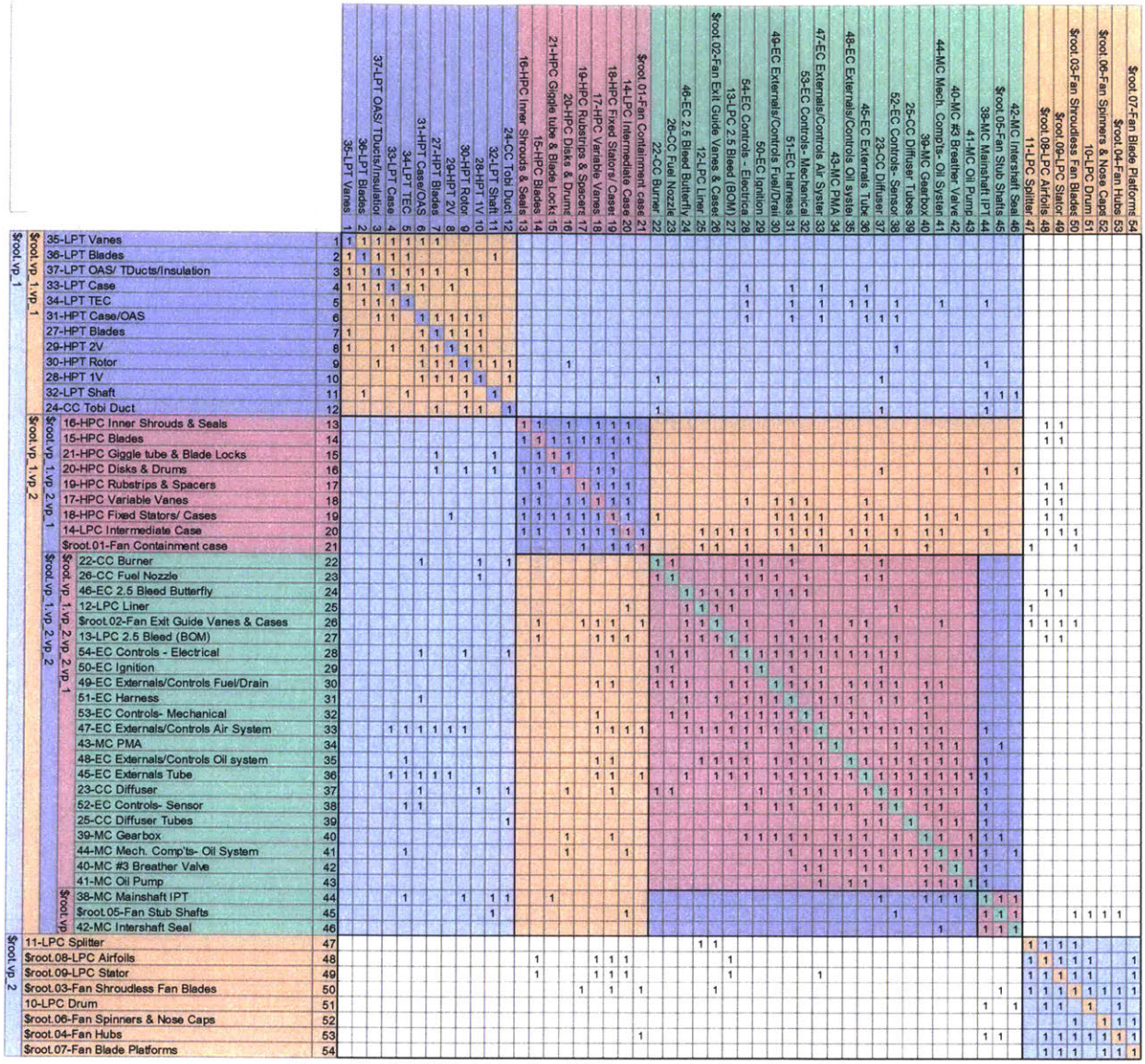


Figure 41- PW4098 Jet Engine DSM re-clustered into 5 modules

Scoring the frequency transform of the 5-module configuration yields an overall modular score of 8.51E-05 and an average intra modular score of 0.047. There are 205 external dependencies outside of the 5 modules in this configuration.

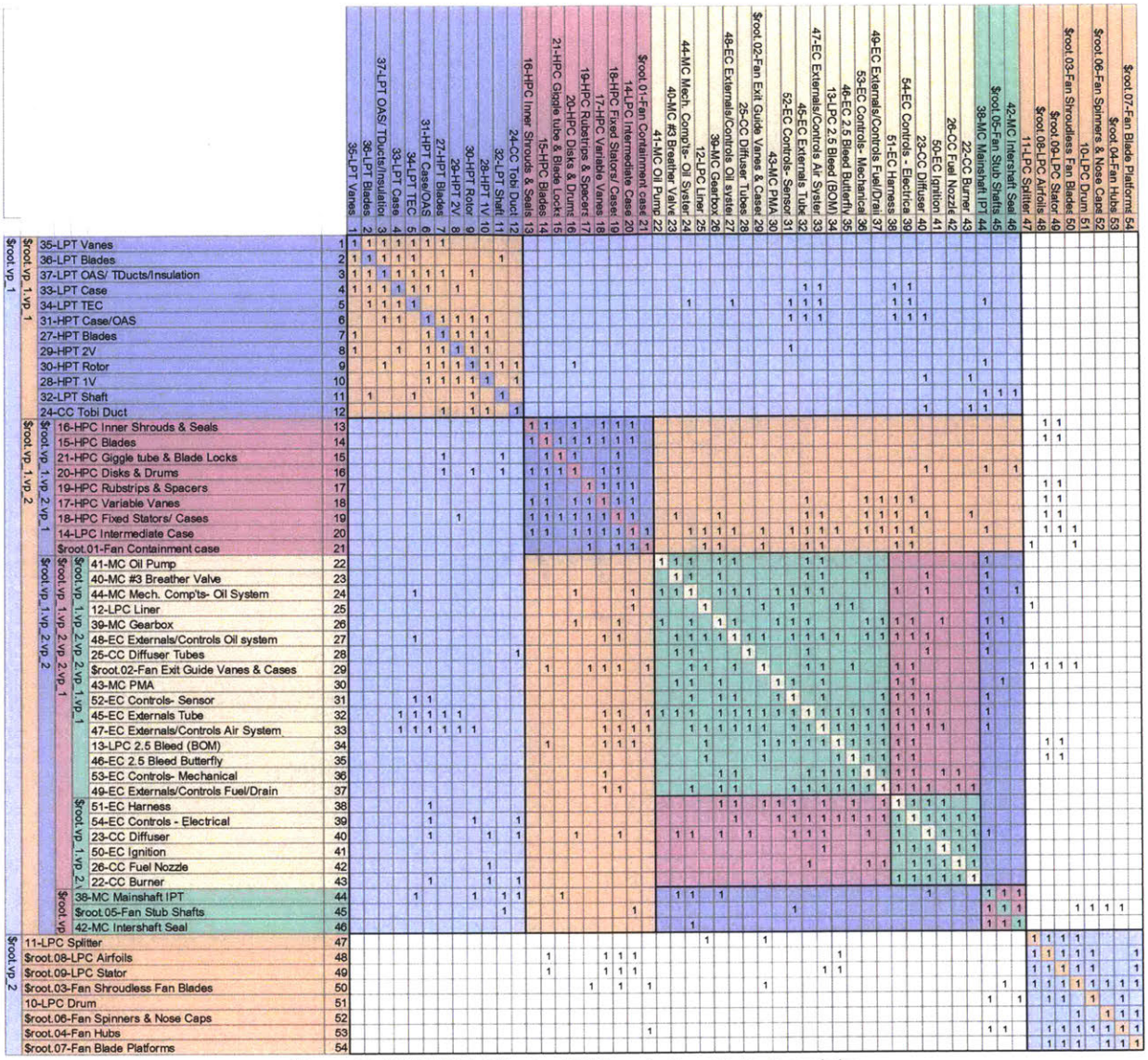


Figure 42-PW4098 Jet Engine DSM re-clustered into 6 modules

Scoring the frequency transform of the 6-module configuration yields an overall modular score of 8.52E-05 and an average intra-modular score of 0.0547. There are 271 external dependencies outside of the 6 modules in this configuration.

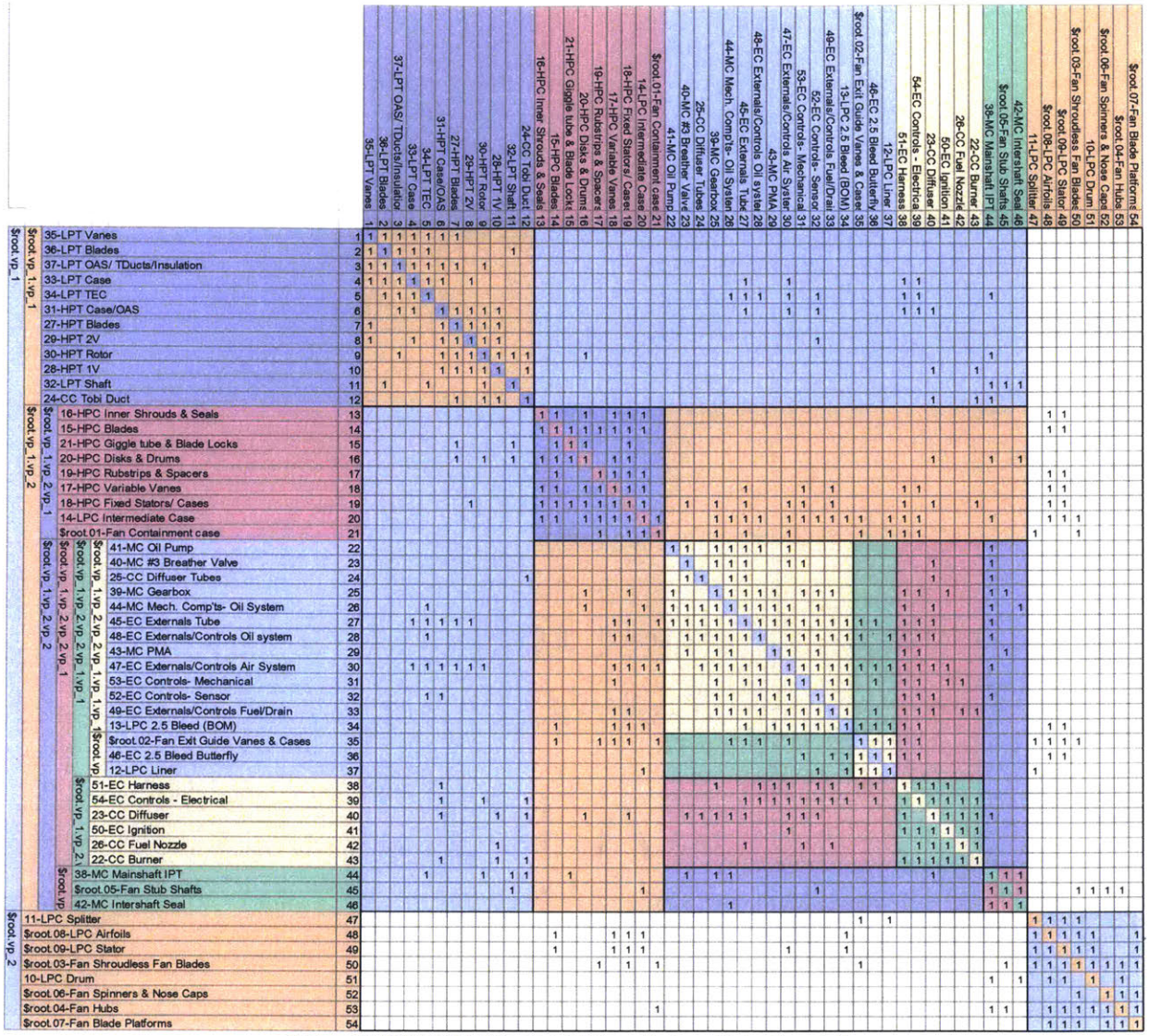


Figure 43-PW4098 Jet Engine DSM re-clustered into 7 modules

Scoring the frequency transform of the 7-module configuration yields an overall modular score of 8.71E-05 and an average intra-modular score of 0.0945. There are 292 external dependencies outside of the 7 modules in this configuration.

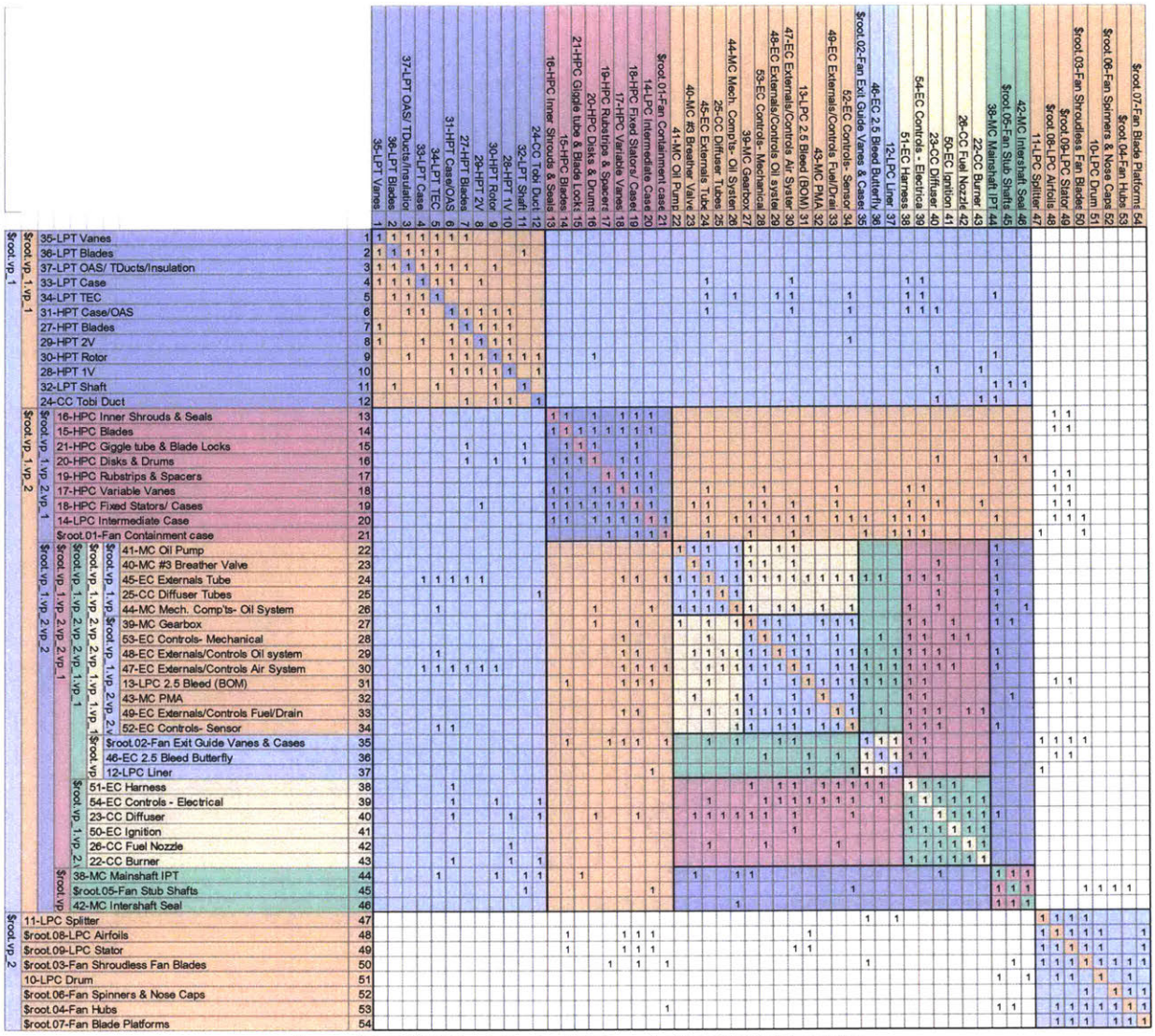


Figure 44-PW4098 Jet Engine DSM re-clustered into 8 modules

Scoring the frequency transform of the 8-module configuration yields an overall modular score of 8.72E-05 and an average intra-modular score of 0.107. There are 328 external dependencies outside of the 8 modules in this configuration.

The results from the scoring are summarized in the table below:

	Modules	Overall score	Intra-module score	External dependencies
Spectral (Normalized) 1 iteration	2	8.11E-05	0.00370	51
Spectral (Normalized) 2 iterations	3	8.00E-05	0.00488	118
Spectral (Normalized) 3 iterations	4	8.71E-05	0.00740	186
Spectral (Normalized) 4 iterations	5	8.51E-05	0.0469	205
Spectral (Normalized) 5 iterations	6	8.52E-05	0.0547	271
Spectral (Normalized) 6 iterations	7	8.71E-05	0.0945	292
Spectral (Normalized) 7 iterations	8	8.72E-05	0.107	328

Anchor (Original architecture)	8	8.87E-05	0.0519	338

Table 3- Summarizing results of scoring the various configurations

When running the spectral (normalized) algorithm iteratively to create more modules, we see a trend of the intra-modular score increasing as well. This is due to the decreasing module sizes, leading to a decreasing number of dependencies encapsulated in each module. The number of external dependencies also increase as these dependencies are no longer encompassed within the modules.

On the 7th iteration of running the spectral (normalized) algorithm, we achieve an 8-module configuration, the same number of modules as the anchor. We see that whilst the anchor architecture has a slightly larger number of external dependencies, the average intra-modular score is lower, whilst the overall modularity is slightly higher.

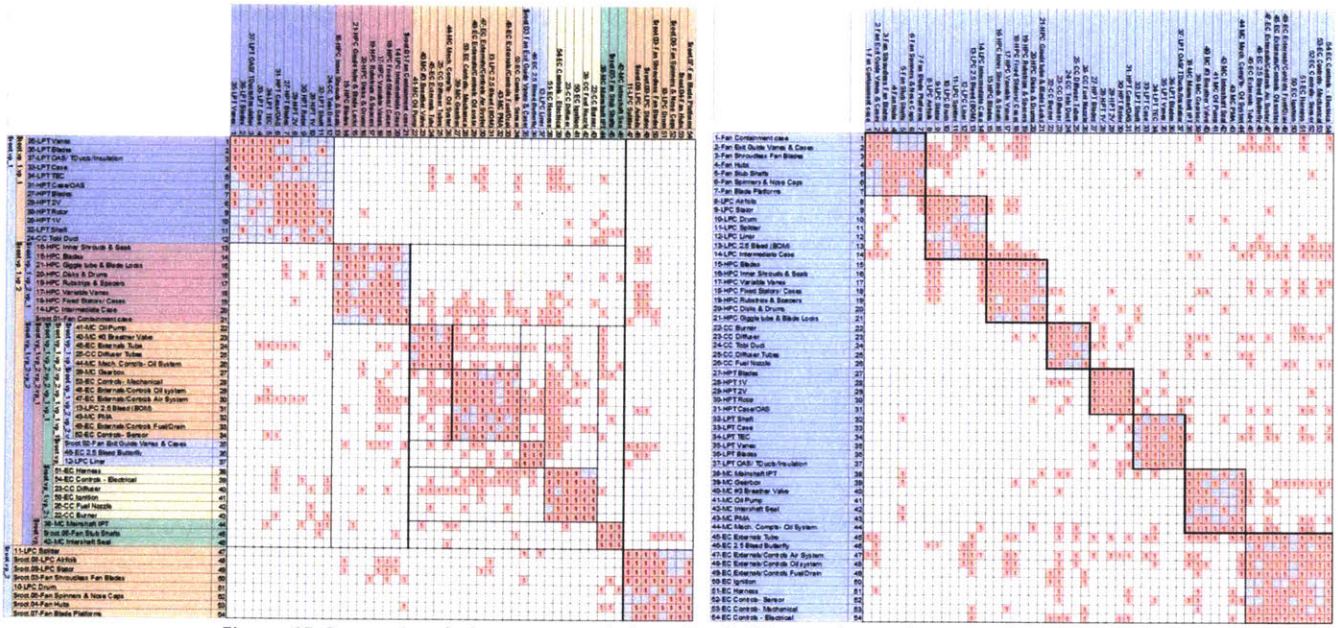


Figure 45-Comparison between spectral clustering (left) vs anchor architecture (right)

There are 328 external dependencies captured in the 8-module spectral-clustered configuration and 338 external dependencies captured in the anchor architecture. While this number is quite close to one another, there is a big difference as to how they are distributed for each case. While the spectral architecture has the external dependencies distributed in a concentric pattern around the middle area, the anchor architecture has its dependencies all concentrated on the bottom and right of the DSM (where the bus is located). As the bus configuration of the anchor architecture shifts most of the dependencies to the sides of the DSM, it is easier for the architect to modularize by inspection.

The higher overall modularity score of the anchor architecture is likely due to the fact that most of the external dependencies are found already grouped together among the bus modules,

resulting in a better score as compared to a configuration where the external dependencies are spread out throughout the system.

The upside of the spectral-clustered configuration is that it has a better average intra-modularity score compared to the anchor architecture. This means that each module, on average exhibits more of the behaviors discussed in the earlier part of the thesis such as banding around the center diagonal or being more compact.

Plotting the number of external dependencies against intra-module modularity in a tradespace, we can see a pattern emerge of increasing intra-module modularity as the number of external dependencies increase:

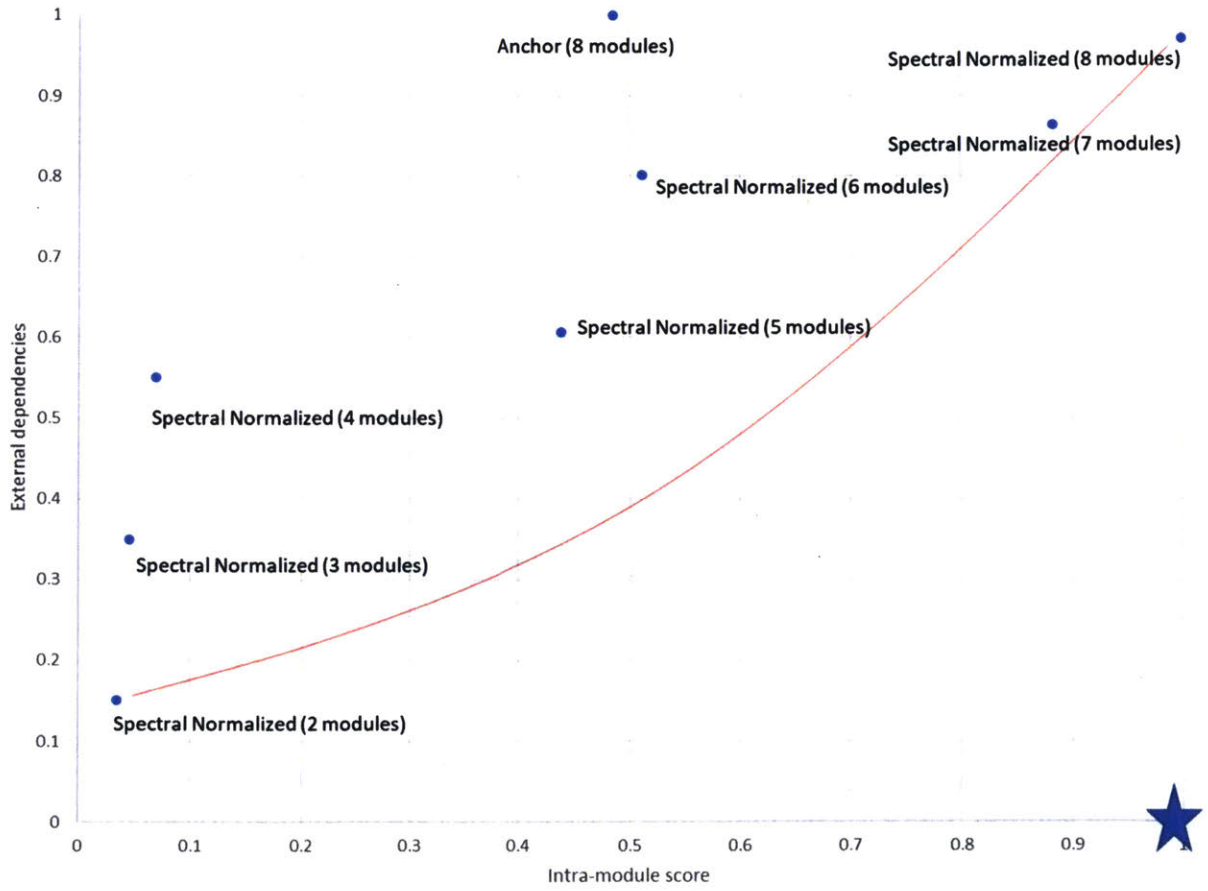


Figure 46- Tradespace containing the various configurations

In the above tradespace, we normalize both the external dependencies and intra-module score to the largest value (which results in the highest score for each being 1). The star on the bottom right corner represents the utopia point where the external dependencies are lowest (best system modularity) and intra- module score is highest (best intra-module modularity). The red

line represents the pareto front where 3 of the configurations are located on. In this tradespace, the stakeholders have an equal tendency toward system modularity and intra-module modularity normalized by the available configurations. He or she would likely pick the spectral normalized (7 module) configuration as it is the closest to the utopia point.

The stakeholder's position

As discussed in the previous section, rather than having an equal tendency toward system modularity and intra-module modularity, stakeholders may instead have a whole spectrum of needs and harbor a preference toward either system modularity or intra-module modularity.

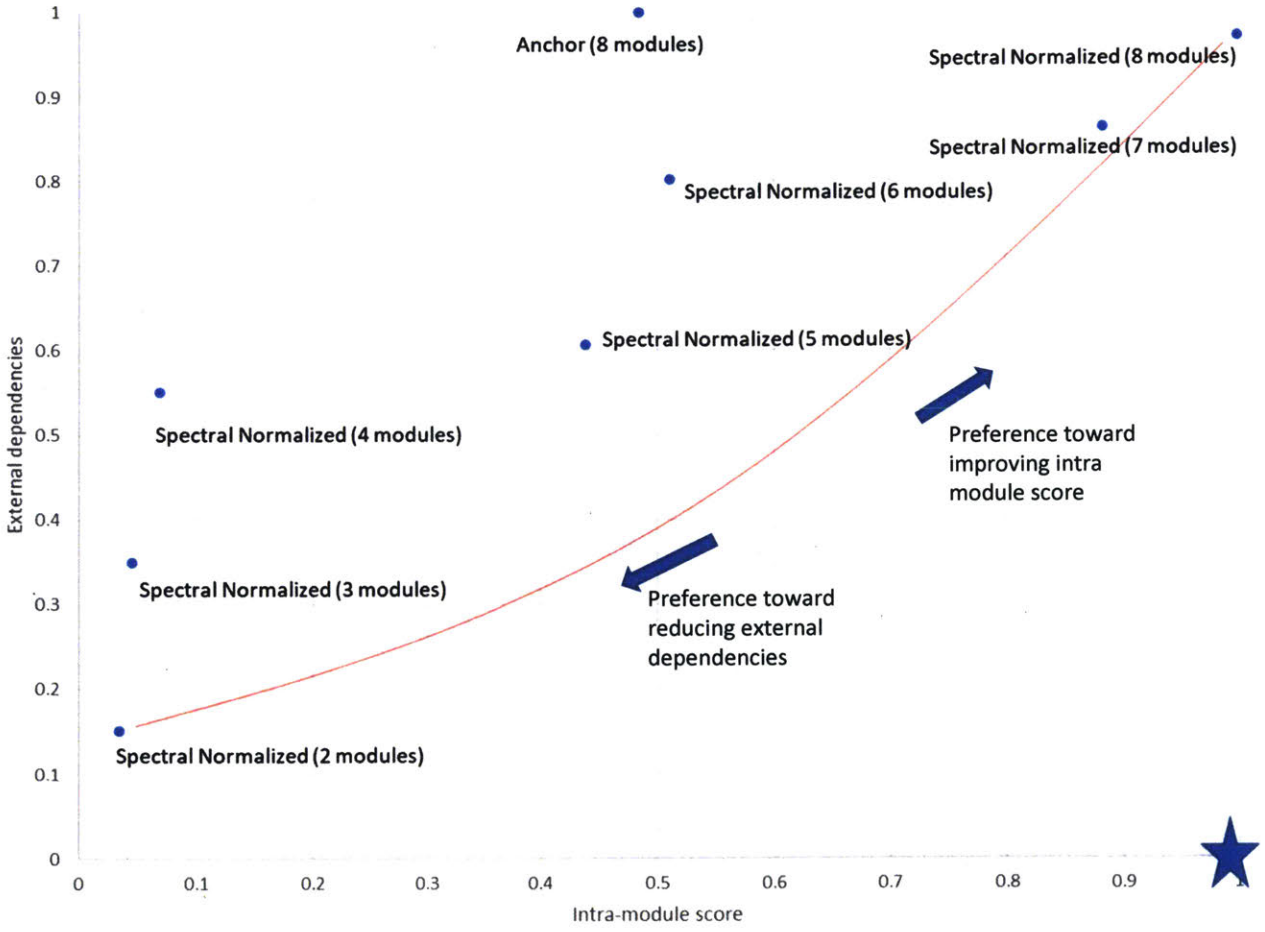


Figure 47- Showing how stakeholder preference affects the preferred configuration on the tradespace

If the stakeholder has a preference toward reducing external dependencies, he would move down along the pareto front and choose the 2-module configuration if the preference is great enough. On the other hand, if the stakeholder has a great enough preference to improve intra-module score, he would move up the pareto front and choose the 8-module configuration.

An alternative way of viewing this would be to consider the position of the utopia point with reference to the configuration's position. If there is a preference toward intra-module score (eg. Intra module score is valued 10 times more than reducing external dependencies) we would divide all the external dependency scores by 10. This would result in the tradespace below:

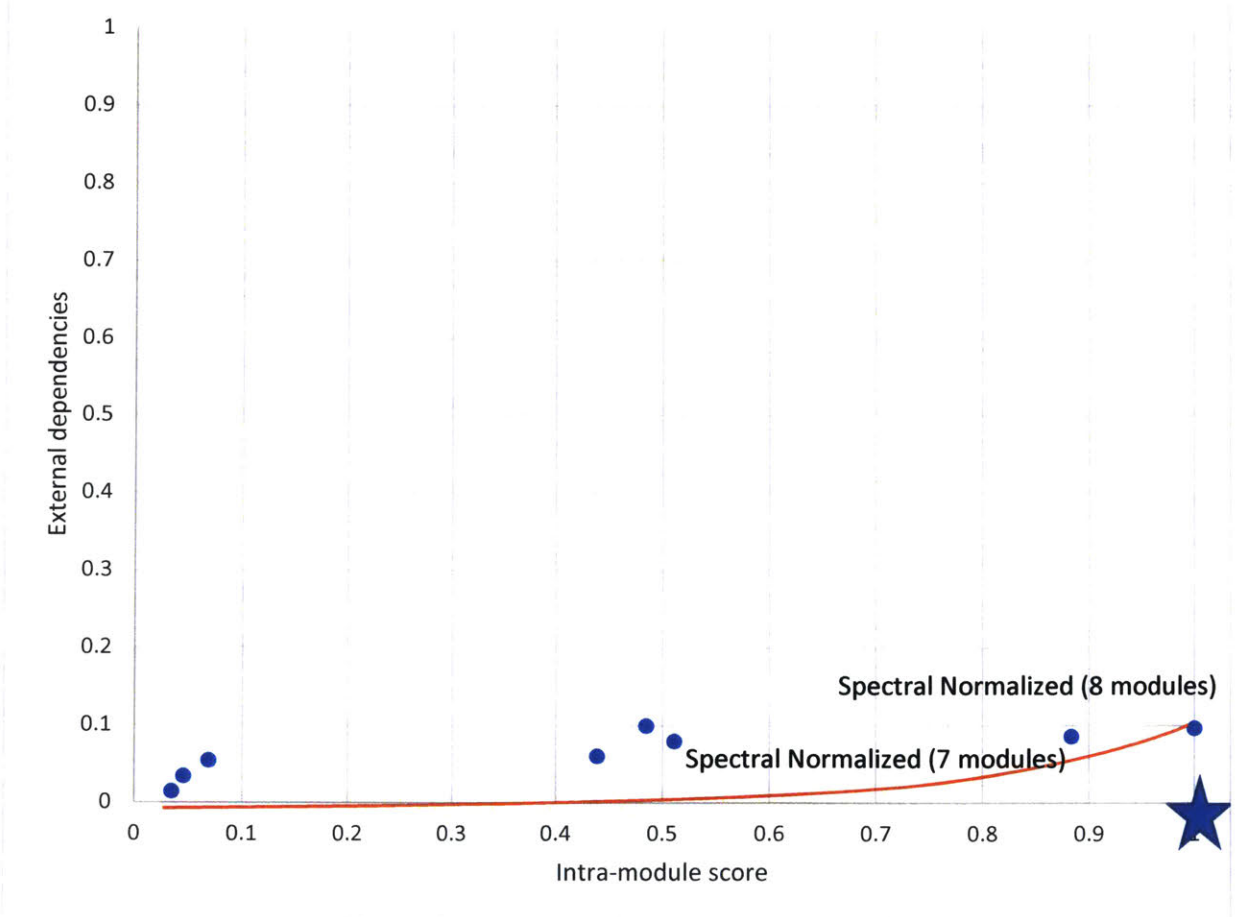


Figure 48- Showing how preference toward improving intra-modular score affects the tradespace

In this case, the 8-module architecture is now the closest to the utopia point, and the stakeholder would likely go with this architecture.

Conversely, if there is a preference toward reducing external dependencies as compared to improving intra-modular score (eg. Reducing external dependencies is valued 10 times more than improving intra-modular score), we would divide all the intra-modular scores by 10, resulting in the tradespace below:

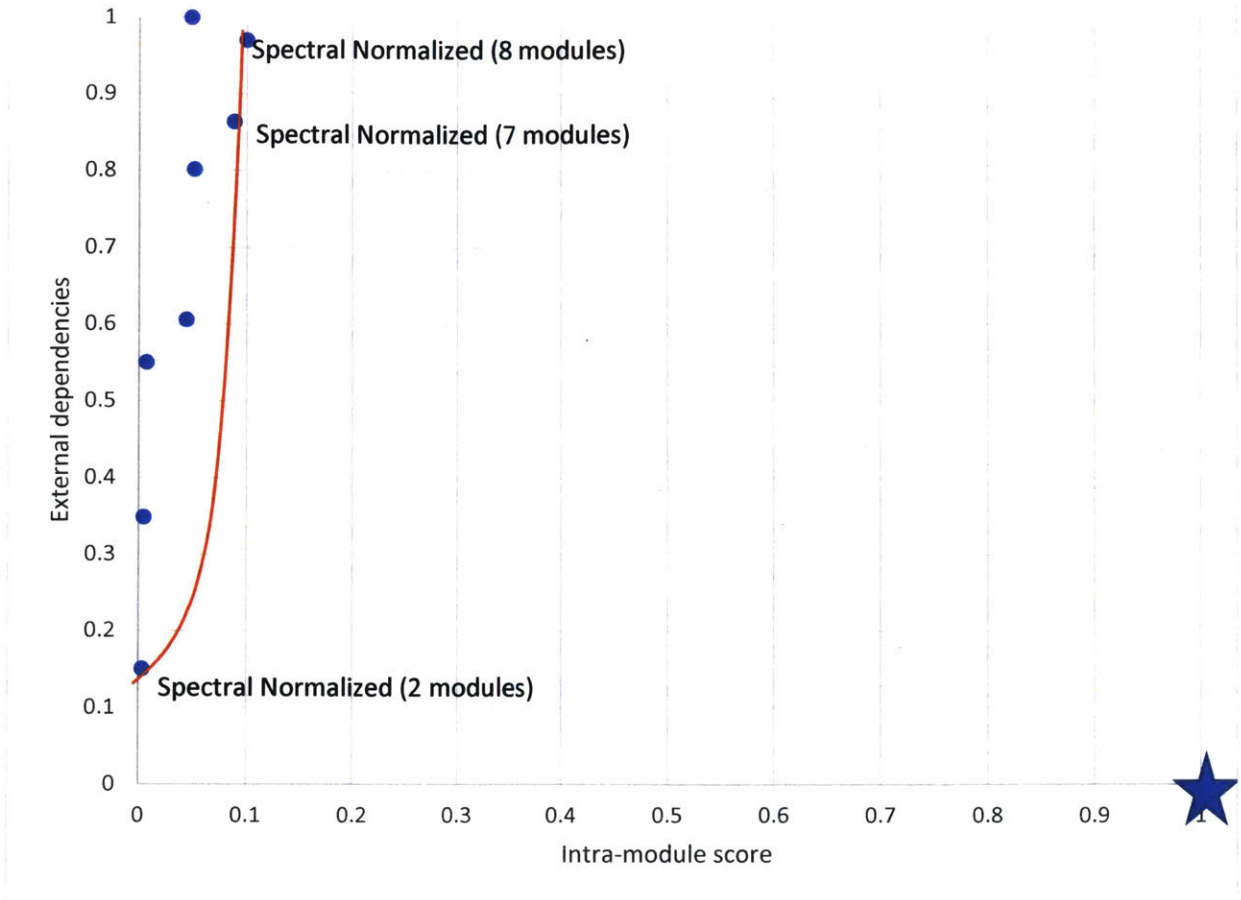


Figure 49- Showing how preference toward reducing external dependencies affect the tradespace

Now, we see that the 2-module configuration is the closest to utopia point, so the stakeholder with a tenfold preference toward reducing external dependencies would likely choose this configuration.

8. Conclusion

In this thesis, we examined the benefits of modularity in the system, and a few ways to improve modularity of a system such as clustering or sequencing. To give a different take on the approach to improving the modularity of a system, we applied an abductive approach by using the Discrete Cosine Transform to look at the frequency components of a DSM and identified patterns in these components which indicated modularity in a system. We then used these patterns to score various configurations to identify useful architectures. Finally, we used the same scoring system as a heuristic to score the modularity of various modules within a system and discussed on the balance between system modularity and intra-module modularity.

Using a DCT to cluster or measure modularity is not meant to be a 'be-all and end-all' to the subject. Rather, it is a tool which can be added to a system architect's repertoire of other transforms, algorithms and metrics which can improve his or her understanding of a particular system. This algorithm could be particularly useful if the architect has some notion of what the end product needs to look like without wanting to force a fit, due to the best effort nature of the frequency based scoring. Another scenario where this algorithm shines is if the architect requires a tradespace of similar configurations which best match his objective, allowing him or her to choose among a number of high scoring configurations.

Future work in this area could include use of DSMs which have run through the algorithm to update and improve the scoring matrix, thus integrating an aspect of machine learning to the way the algorithm performs. As only sequencing and simple clustering have been covered in this paper, exploration of scoring matrixes which can cover other characteristic behaviors can be another area of improvement. Finally, as this paper covers only DSMs with unweighted dependencies, the behavior and patterns of weighted dependencies in the frequency domain can be another area which can be looked at.

Bibliography

[1] B. Sandeep Sovani, "Addressing Engineering Challenges of Increasingly Complex Automobiles."

[2] "The concept of project complexity—a review," *Int. J. Proj. Manag.*, vol. 14, no. 4, pp. 201–204, Aug. 1996.

[3] D. J. Sturtevant, "System design and the cost of architectural complexity," 2013.

[4] D. E. Whitney, "PHYSICAL LIMITS TO MODULARITY," 2004.

[5] K. B. Clark *et al.*, "Modularity in the Design of Complex Engineering Systems," 2004.

[6] R. Garud and A. Kumaraswamy, "Technological and organizational designs for realizing economies of substitution," *Strateg. Manag. J.*, vol. 16, no. S1, pp. 93–109, Jan. 1995.

[7] T. R. Browning, "Applying the Design Structure Matrix to System Decomposition and Integration Problems: A Review and New Directions," *IEEE Trans. Eng. Manag.*, vol. 48, no. 3, 2001.

[8] S. D. Eppinger and T. R. Browning, *Design structure matrix methods and applications*. MIT Press, 2012.

[9] R. I. Whitfield, J. S. Smith, and A. H. B. Duffy, "IDENTIFYING COMPONENT MODULES," pp. 15–17, 2002.

[10] J. N. Warfield, "Binary Matrices in System Modeling," *IEEE Trans. Syst. Man. Cybern.*, vol. SMC-3, no. 5, pp. 441–449, Sep. 1973.

[11] R. E. Thebeau, "Knowledge Management of System Interfaces and Interactions for Product Development Processes," 2001.

[12] J. Idicula, "Planning for Concurrent Engineering," 1995.

[13] C. I. Gutierrez, "Integration analysis of product architecture to support effective team co-location," 1998.

[14] "DSMweb.org: Sequencing a DSM." [Online]. Available: http://www.dsmweb.org/en/understand-dsm/technical-dsm-tutorial/partitioning.html. [Accessed: 19-Dec-2017].

[15] I. Cook and G. Coates, "Optimising the time-based design structure matrix using a divide and hybridise algorithm," *J. Eng. Des.*, vol. 27, no. 4–6, pp. 306–332, May 2016.

[16] S.-H. 1975- Cho, "An integrated method for managing complex engineering projects using the design structure matrix and advanced simulation," 2001.

[17] Y. C. Ho, "Abduction? Deduction? Induction? Is there a Logic of Exploratory Data Analysis?," 1994.

[18] N. J. Pioch, D. Hunter, J. V White, A. Kao, D. Bostwick, and E. K. Jones, "Multi-Hypothesis Abductive Reasoning for Link Discovery."

[19] S. A. Khayam, "The Discrete Cosine Transform (DCT): Theory and Application," *Components*, 2003.

[20] E. Crawley, B. Cameron, and D. Selva, *System architecture : strategy and product development for complex systems*. .

[21] D. Verma and M. Meila, "A comparison of spectral clustering algorithms," *Univ. Washington, Tech. Rep. UW-CSE-03-05-01*, 2003.

[22] L. Hagen and A. B. Kahng, "New spectral methods for ratio cut partitioning and clustering," *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 11, no. 9, pp. 1074–1085, 1992.