Acquiring Symbolic Design Optimization Problem Reformulation Knowledge

On computable relationships between design syntax and semantics

Somwrita Sarkar

A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

February 2009

Design Lab
Faculty of Architecture, Design and Planning
University of Sydney

© Somwrita Sarkar
To my parents
ABSTRACT

Acquiring Symbolic Design Optimization Problem Reformulation Knowledge:
On computable relationships between design syntax and semantics

This thesis presents a computational method for the inductive inference of explicit and implicit semantic design knowledge from the symbolic-mathematical syntax of design formulations using an unsupervised pattern recognition and extraction approach. Existing research shows that AI / machine learning based design computation approaches either require high levels of knowledge engineering or large training databases to acquire problem reformulation knowledge. The method presented in this thesis addresses these methodological limitations. The thesis develops, tests, and evaluates ways in which the method may be employed for design problem reformulation.

The method is based on the linear algebra based factorization method Singular Value Decomposition (SVD), dimensionality reduction and similarity measurement through unsupervised clustering. The method calculates linear approximations of the associative patterns of symbol co-occurrences in a design problem representation to infer induced coupling strengths between variables, constraints and system components. Unsupervised clustering of these approximations is used to identify useful reformulations. These two components of the method automate a range of reformulation tasks that have traditionally required different solution algorithms. Example reformulation tasks that it performs include selection of linked design variables, parameters and constraints, design decomposition, modularity and integrative systems analysis, heuristically aiding design “case” identification, topology modeling and layout planning.

The relationship between the syntax of design representation and the encoded semantic meaning is an open design theory research question. Based on the results of the method, the thesis presents a set of theoretical postulates on computable relationships between design syntax and semantics. The postulates relate the performance of the method with empirical findings and theoretical insights provided by cognitive neuroscience and cognitive science on how the human mind engages in symbol processing and the resulting capacities inherent in symbolic representational systems to encode “meaning”. The performance of the method suggests that semantic “meaning” is a higher order, global phenomenon that lies distributed in the design representation in explicit and implicit ways. A one-to-one local mapping between a design symbol and its meaning, a largely prevalent approach adopted by many AI and learning algorithms, may not be sufficient to capture and represent this meaning. By changing the theoretical standpoint on how a “symbol” is defined in design representations, it was possible to use a simple set of mathematical ideas to perform unsupervised inductive inference of knowledge in a knowledge-lean and training-lean manner, for a knowledge domain that traditionally relies on “giving” the system complex design domain and task knowledge for performing the same set of tasks.
ACKNOWLEDGEMENTS

I thank my thesis supervisor, Dr. Andy Dong, and associate supervisor, Prof. John Gero, for the many happy hours of learning. Andy taught me how to be simultaneously rigorous and adventurous while thinking about ideas. Every aspect of this thesis has benefited from discussions with Andy. I hope to become a teacher someday, and I take home many valuable lessons from Andy on how to be an inspiring teacher, mentor and friend all at once. John taught me how never to lose “the big picture” while investigating the tiniest detail, nook or corner. I came to Sydney University with almost no background in AI. I thank John for introducing me to a whole world of new and challenging ideas, for the faith that I will be able to pursue a PhD in a topic that I was to learn from scratch, and for all the help and support that I could always ask for and receive in the process.

Thanks are due to all the faculty members of the Design Lab, and to all my fellow Mezzanine members. It has been a great pleasure to spend many interesting hours with all. I thank Rob Saunders for his insights and discussions while this thesis was in development. Vishal has been the great friend and ear for “back home” talks. Nick was the music pal. Lucila and Fiona gave me all the tips that new mums doing PhDs should know.

Last, and most important, I thank my family. If there is anyone who derives more happiness from this PhD than me, it is Baba and Ma, my parents. To them is due the childhood lesson of being simple, curious and enthusiastic about everything in the world, a lesson that continues to grow through the years by example. I thank my brother for those unexpected merry phone calls night or day whenever thesis times got heavy. I thank my grandmother for her immeasurable love that always reaches me despite the distance. I thank Chacha, Chachi and Mummy for their love and support. I thank Ritu, Neesha and Mukta for being the best friends one could ever have – trivial matters like space and time, living in different corners of the world or meeting once in years, never cause any distance or separation.

I thank my husband, Mayank, for being my best buddy through the years, for the laughter, fights and togetherness, and above all, for all the caring and loving that no words can amply describe through the best and worst of thesis times. Kaivalya, our son, turns one as I submit this thesis. He is my best source of inspiration. After all, arriving into a new world, and learning about it with unconditional enthusiasm and sparkling eyes, after many falls, is much harder than doing a PhD.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Occurrence matrix</td>
</tr>
<tr>
<td>A'</td>
<td>$k$-reduced occurrence matrix</td>
</tr>
<tr>
<td>E</td>
<td>$SV^T$</td>
</tr>
<tr>
<td>f</td>
<td>Vector of objective functions in analytical design optimization problems</td>
</tr>
<tr>
<td>g</td>
<td>Vector of inequality constraints in analytical design optimization problems</td>
</tr>
<tr>
<td>h</td>
<td>Vector of equality constraints in analytical design optimization problems</td>
</tr>
<tr>
<td>k</td>
<td>Number of dimensions retained</td>
</tr>
<tr>
<td>m</td>
<td>Number of rows in $A$</td>
</tr>
<tr>
<td>n</td>
<td>Number of columns in $A$</td>
</tr>
<tr>
<td>O</td>
<td>$US$</td>
</tr>
<tr>
<td>p</td>
<td>Vector of parameters in analytical design optimization problems</td>
</tr>
<tr>
<td>r</td>
<td>Rank of $A$</td>
</tr>
<tr>
<td>$R^n$</td>
<td>$n$-dimensional real space</td>
</tr>
<tr>
<td>S</td>
<td>Matrix of singular values in SVD</td>
</tr>
<tr>
<td>U</td>
<td>Matrix of left singular vectors in SVD</td>
</tr>
<tr>
<td>V</td>
<td>Matrix of right singular vectors in SVD</td>
</tr>
<tr>
<td>x</td>
<td>Vector of design variables in analytical design optimization problems</td>
</tr>
<tr>
<td>$X(k)$</td>
<td>Event-event cosine similarity matrix</td>
</tr>
<tr>
<td>$Y(k)$</td>
<td>Episode-episode cosine similarity matrix</td>
</tr>
<tr>
<td>$Z(k)$</td>
<td>Event-episode cosine similarity matrix</td>
</tr>
</tbody>
</table>
Publication notes


Table of Contents

Abstract ................................................................................................................................. iii
Acknowledgements .............................................................................................................. iv
List of Symbols ..................................................................................................................... v
Publication notes .................................................................................................................. vi

Chapter 1. A Semantic – Syntactic Puzzle ..................................................................... 1
  1.1 Motivation ..................................................................................................................... 2
  1.1.1 Design methodology motivation ........................................................................... 3
  1.1.2 Design theory motivation ....................................................................................... 5
  1.2 Aim .............................................................................................................................. 6
  1.3 Objectives .................................................................................................................... 7
  1.4 Research claims, contributions and significance ....................................................... 7
  1.4.1 Design methodology ............................................................................................ 7
  1.4.2 Design theory ...................................................................................................... 10
  1.5 Thesis structure ......................................................................................................... 11

Chapter 2. An Analogy Based Solution: Semantics from Syntax .............................. 13
  2.1 Review of existing approaches: establishing behavior criteria ............................... 14
  2.2 Approach used in this thesis ....................................................................................... 17
  2.3 Analogies .................................................................................................................... 20
  2.3.1 Structural analogy .............................................................................................. 21
  2.3.2 A common representation form from the structural analogy ............................... 24
  2.4 Behavioral analogy ................................................................................................... 24
  2.4.1 What does SVD do? .......................................................................................... 25
  2.4.2 What does SVD do in LSA? .............................................................................. 26
  2.4.3 What does SVD do in DIP? .............................................................................. 28
  2.5 Behavior characteristics from SVD, dimensionality reduction, similarity measurement 29
  2.5.1 Measure of “semantic similarity” in design representations .............................. 29
  2.5.2 Dimensionality reduction and implicit pattern extraction ................................... 30
  2.5.3 Redundancy, matrix compression and explicit pattern extraction ....................... 31
  2.5.4 Multiple and invariant patterns from a single data set: ...................................... 32
  2.6 Summary .................................................................................................................... 33

Chapter 3. Theoretical Conjectures: Why Does it Work? ........................................... 34
3.1 What is the connection between design syntax and semantics? ....................................... 35
3.2 Physical Symbol System Hypothesis: Symbols are [semantic] knowledge ..................... 36
3.3 Insights from cognitive science and cognitive neuroscience............................................ 37
  3.3.1 Situated cognition: Perceptual-conceptual couplings re-coordinate to produce a symbol......................................................................................................................... 37
  3.3.2 “Symbols aren’t simple”: Why a purely syntactical analysis reveals “meaning” ..... 38
  3.3.3 Acquiring and inferring knowledge from episodes.................................................... 41
3.4 Summary: Additional performance criteria for method ................................................... 42

Chapter 4. Method: REIFORM ................................................................................................. 44
  4.1 Step I: Data Representation .............................................................................................. 45
  4.1.1 Analytical formulations ............................................................................................. 45
  4.1.2 Non-analytical formulations ...................................................................................... 46
  4.2 Step II: Performing SVD on the occurrence matrix ......................................................... 48
  4.2.1 Defining the event (variable, parameter, design component) space ......................... 48
  4.2.2 Defining the episode (function/interaction) space ..................................................... 49
  4.2.3 Defining the US and SV^T spaces ............................................................................. 50
  4.2.4 Geometric interpretation of mathematics ................................................................ 50
  4.3 Step III: Dimensionality reduction .................................................................................. 51
  4.3.1 Interpreting approximations to US and SV^T spaces: .................................................. 51
  4.3.2 How do the approximations capture implicit information? ....................................... 52
  4.3.3 Example demonstration ............................................................................................. 53
  4.3.4 How implicit relationships change over k values ...................................................... 54
  4.4 Vector representation of dimensionality reduction: geometric interpretation .............. 56
  4.5 Step IV: Similarity measurement ..................................................................................... 57
  4.5.1 Cosine similarity measurement.................................................................................. 58
  4.5.2 Unsupervised K-means clustering ............................................................................. 61
  4.6 A note on computational implementation ........................................................................ 62
  4.7 Summary .......................................................................................................................... 62

Chapter 5. Design Decomposition and Modularity Analysis ............................................. 64
  5.1 Design decomposition ...................................................................................................... 64
  5.1.1 Existing research ....................................................................................................... 65
  5.1.2 Using REIFORM for decomposition ......................................................................... 67
  5.1.3 Method for decomposition ......................................................................................... 68
  5.1.4 Matrix partitioning approach for decomposition ...................................................... 69
Chapter 10. Conclusions and Future Work .............................. 127

10.1 Summary .......................................................................................... 127
10.2 Review of claims ...................................................................................... 128
  10.2.1 Knowledge lean ................................................................................. 128
  10.2.2 Training lean ...................................................................................... 129
  10.2.3 Generality .......................................................................................... 130
  10.2.4 Capturing multiplicity and invariance of design knowledge ............... 131
10.3 Future work ......................................................................................... 133
  10.3.1 Method extensions .............................................................................. 133
  10.3.2 Method application to new areas .......................................................... 135
  10.3.3 Design theory: “Global” design computation methods and the role of statistical, unsupervised methods ................................................................................................................. 138
10.4 In the end ......................................................................................... 139

References ................................................................. 140

Appendix A ................................................................. 143
Chapter 1

A Semantic – Syntactic Puzzle

“When I use a word,” Humpty Dumpty said in a rather scornful tone, “it means just what I choose it to mean…neither more nor less.” “The question is,” said Alice, “whether you CAN make words mean so many different things.” “The question is,” said Humpty Dumpty, “which is to be master, that’s all.”

Through the Looking Glass, Lewis Carroll

The development of AI algorithms to automate or support symbolic design problem reformulation is an enduring challenge in design computation. Existing research shows that design tools either require high levels of knowledge engineering or large databases of training cases. To address some of these limitations, this thesis presents REIFORM, a method based on the linear algebra factorization method of Singular Value Decomposition (SVD), dimensionality reduction, similarity measurement, and unsupervised clustering. REIFORM performs design reformulation by inductively acquiring design knowledge from symbolic design representations. Using patterns of associations that exist between symbols, it uncovers global, implicit knowledge of the design problem from the local, explicit information in the problem representation. This ability scopes the kind of reformulation tasks that it can perform – those in which a problem needs to be “seen” as decomposed into coupled or interacting substructures in the process of reformulating it (the meaning of “decomposition” is used in its most general conceptual sense here). Example reformulation tasks that it performs include

1 Decomposition is used in two “senses” in this thesis. In the conventional sense, decomposition implies a direct problem reformulation task, i.e. breaking up a design problem or system into its component parts. In a broader conceptual sense, it implies that any design problem or system can be “seen” in terms of interacting, coupled sub-structures, for any problem reformulation task, whether or not the final objective may be to decompose the system.
selection of linked design variables, parameters and constraints, design decomposition analysis, modularity and integration analysis, heuristically aiding design “case” identification, and constraint satisfaction tasks such as topology modeling and layout planning. The name REIFORM derives from the words “reification” and “reformulation”, motivated by the observation that design modeling and reformulation are acts of symbolic reification. For clarity, the word REIFORM will be used in place of “the method for design problem modeling and reformulation presented in this thesis”.

1.1 Motivation

Knowledge and its representation – we define, divide, arrange and organize knowledge, reifying it into categories, into subjects and disciplines of inquiry, into symbolic systems of representation. In contrast, the world, and our existence in the world, is messy, changing, shifting, non-classified. In our attempts to make sense of it all, we extract from this mass of tangled entities, relationships and associations, a few apparent patterns through “this” spatial filter or “that” temporal lens defined by some particular, personal goal. We give names to them and live on the assumption that these few extracted patterns, representative of one interpretive, subjective viewpoint in particular moments in time, represent the entire network.

Much, however, lies unrepresented. A representation is necessarily more limited than the thing it represents, because representation, by definition, is abstraction. Yet, human beings learn, infer and construct knowledge of the world by employing these abstractions through representations.

Design is a discipline with the pragmatic aim of transforming the world we inhabit into a more desirable future world (Simon, 1969/ 1981). Design deals with a vision that has to be described and represented in order to be realized. This act of describing is a challenge because what is being described does not exist yet. Design modeling and reformulation involve reifying the semantic meaning of a design work (as conceived by the designer) into a formal representation. In the modeling phase, design knowledge is known to be ill-defined, ill-structured and incomplete (Simon, 1969/ 1981). To represent such knowledge in formal symbolic-mathematical terms is a challenge, because mathematics is a well-structured and unambiguous knowledge domain and requires precision in expression. The physical realization of any design work enacts the tension between the need to reason with ill-structured knowledge of the world and the need to represent such knowledge in a well-structured manner. It is during the modeling phase that the semantic-syntactic relationships between design knowledge and its representation are the most unclear and subject to change. Yet, it is at this stage that the levels of abstraction are formalized and a representation created.
One reason why AI / machine learning face a challenge in automating problem modeling and reformulation is because the cognitive mechanisms by which designers do these tasks are not well-understood, and hence cannot be computationally modeled in an exhaustive manner. Instead of focusing on the cognitive mechanisms, therefore, in this research we turn to the symbolic system produced by the cognitive mechanisms – the syntax of design representation. Because the design representation is the result of cognitive processes, it must encode the semantics. Thus, it should be possible to acquire the semantics from the syntax and use it for problem reformulation.

This thesis focuses on symbolic-mathematical problem modeling and reformulation in design. For the purpose of this thesis, “design syntax” refers to a symbolic or mathematical language that is employed to represent a design problem. “Semantic meaning” of a design refers to the structure and behavior of the design object being modeled, i.e. the design elements and relationships as conceived of by a designer. The designer expresses the semantic meaning of a design through a symbolic or mathematical model. The model describes a space of solutions that is searched for a feasible or optimal solution. From a design theory perspective, this thesis addresses the development of computable relationships between the explicit and implicit semantic meaning of a design work and its symbolic-mathematical representation. From a design methodology perspective, the thesis addresses how this computation of semantics from syntax forms the basis for a design problem reformulation method.

The following definitions and terminology will be used for discussion in the rest of the thesis. In this thesis, a symbolic-mathematical model is considered to be a syntactic encoding of semantic structural-behavioral design knowledge. Such a model can be stated as a general design problem (Michelena & Papalambros, 1997), an optimal design problem (Papalambros & Wilde, 2000), a standard non-analytic Functional Dependence Table (FDT) or a Design Structure Matrix (DSM) (Li & Li, 2005; Pimmler & Eppinger, 1994). A symbol in a design model represents a structure or behavior variable (for e.g. internal diameter, force), parameter (e.g. minimum wall thickness), or system component (e.g. Fan Blade in an aeroengine). Such symbols come together in functional, interaction (e.g., spatial adjacency), or dependency (e.g., causal, flow) relationships. An occurrence matrix is a matrix that captures how symbols come together to define functions and dependency/interaction relationships.

1.1.1 Design methodology motivation

This research commenced with the pragmatic aim of developing a computational method to automate problem modeling and reformulation tasks in symbolic-mathematical design (especially design optimization problem reformulation), motivated by the observation that current design tools do not support the designer in this task.
Design problem reformulation significantly affects the final results of any design optimization process. While a mathematical model of a design problem may contain all the relevant structural and behavioral knowledge of the design work, it may not be well formed for optimization. Therefore, a reformulation of a design problem is any action that enables a designer to convert a design model to a well-behaved form, one that may be solved more easily as a result of this reformulation.

Currently, design optimization tools do not support the designer to reformulate the problem due to the substantial knowledge engineering that would have to be embedded into such a tool. Design problem reformulation is difficult to automate because the task is dependent on interaction with the design problem, knowledge intensive, requires human subjective decision-making and a large amount of domain and mathematical expertise (Papalambros & Wilde, 2000). Designers learn from years of experience and significant trial and error to describe a mathematical model that satisfactorily captures all modeling requirements, and guarantees both the existence of a solution and the computation of an optimal or feasible solution within a bounded computation time through the application of a numeric or symbolic algorithm. In problem reformulation, the same design work may be modeled in multiple ways, depending upon personal, design-based or mathematical choices exercised by the designer. The problem definition evolves as the designer’s understanding of the problem grows. Many reformulations are typically required to reach a satisfactory model, making problem formulation and solution an iterative, cyclical process. There is no known formal (cognitive or computational) process that takes an abstract set of modeling and design requirements as input and produces a symbolic-mathematical design representation as output.

Due to these and many other related reasons, problem modeling and reformulation remains, largely, a human endeavor. There are many search based numerical or symbolic algorithms that can find optimal or feasible solutions once a model is defined. However, there are few algorithms that allow a designer to explore modeling and reformulation possibilities. As the complexity of processes and products in engineering increases, design models in optimization become cognitively intractable in terms of the number of variables, parameters, objectives and constraints and their interactions. The dimensionality of the problem often goes beyond human short-term memory capacities. There is a need for computational systems to provide support for tasks that were previously considered purely in the human domain. From a pragmatic design methodology and practice standpoint, it would be beneficial to develop computational methods that could assist the designer with problem modeling and reformulation tasks.
1.1.2 Design theory motivation

The physical symbol system hypothesis (Newell & Simon, 1976; Simon, 1995) establishes a prevalent methodological and theoretical approach for developing AI algorithms. This is the explicit symbolic reasoning approach (Cagan et al., 1997), which specifies that knowledge is explicitly represented in terms of symbols and symbolic processes. This theory has had a major influence on the development and application of AI algorithms in design computation. It is a powerful approach that lies at the basis of many heuristic and exact design computation methods. Rule based expert systems, case based reasoning systems, and general, symbolic mathematics or logic based reasoning systems are all examples of this methodological approach.

In this theory, the semantic “meaning” of a symbol and the relationship of this symbol to all the others are made explicit at the time of defining the knowledge (Newell & Simon, 1976). This fixes the semantic-syntactic mapping. An explicit re-definition is required to change this mapping.

Contrast this theory and its resulting models with the observable cognitive aspects of modeling and reformulation. Problem modeling and reformulation is a phase during which the semantic “meaning” contained by a symbol and relationships between symbols are still in development. In this phase, human designers show a capacity to change the mapping between a symbol and its semantic “meaning” or the “role” played by the symbol in the representation. This allows for a flexible redefinition of knowledge relationships in the specific local context of the particular problem being modeled. As an example, consider a very common design decision that designers take (Papalambros & Wilde, 2000) – whether a certain physical quantity is to be a variable, or is to be kept fixed in a mathematical model (indeed whether it is to be represented at all). If it is kept variable, it becomes a decision variable in the model; if kept fixed, it becomes a parameter. This is often a subjective decision dependent on the designer. For example, the designer may wish to reduce the dimensionality of the problem model, or may wish to study how other variables and constraints behave if one is kept constant. Regardless of these specific details, the important points to note are: (1) at any point in time, the designer may choose to change this semantic “meaning” mapping – a quantity that was previously a parameter may be turned into a variable or vice versa; and, (2) one mapping change for one symbol results in a change in how all the symbols related to this symbol are semantically perceived by the designer. These points hint towards a “global” pattern recognition based inference approach that human designers seem to exhibit. The pattern recognition may be one of many reasons for their robust problem modeling abilities – the ability to flexibly deal with associative relationships that exist between sets of “perceptual” symbols within a design experience. Not only can the same symbol have
differing “meanings” in differing contexts, and different symbols have the same “meaning” in
different contexts depending on the symbol set being used in a particular design experience,
the semantic interrelationships between these symbols within the local context of a single
design experience are dynamic. Local syntactical changes can lead to global meaning or role
changes. This significant ability to treat the syntactic-semantic mapping in a dynamic and
flexible manner is an ability that is missing from many design computation algorithms based
on the explicit symbolic reasoning approach.

This thesis is motivated by findings in statistical natural language processing, cognitive
science and cognitive neuroscience that throw an alternate light on how the human mind
engages in symbol processing and the resulting capacities inherent in symbolic
representational systems to encode “meaning” – views that can provide additional metaphors
and behavior criteria for development of computational methods. The research is motivated to
explore these alternate ways in which symbols may interact to reify semantic design
knowledge beyond what is suggested by the explicit symbolic reasoning approach.

There is no claim that this research is computationally modeling any cognitive
phenomenon. However, as the complexity of processes and products in engineering design
increases, the issue of how a human designer interacts with a computer and how
computational methods support more “intuitive” interaction becomes important. Given the
large and complex nature of qualitative and quantitative domain knowledge involved in
design problem modeling tasks, and the fact that designers themselves do not know the best
formulation in advance, there is a need for methods that can allow a designer to develop
exploratory insights into models. The explicit symbolic reasoning approach is powerful but
requires rules, heuristics, grammar based or functional/qualitative/logical reasoning based
knowledge engineering. This implies that any reformulation of the design model has to be
encoded in terms of explicit sets of mathematical or logical rules. From a design theory
perspective, it would be beneficial to develop additional and alternate models of knowledge
engineering that could support modeling and reformulation in knowledge lean ways.

1.2 Aim

This thesis aims to present a computational method for the inductive inference of explicit and
implicit semantic design knowledge from the symbolic-mathematical syntax of design
formulations using an unsupervised pattern recognition and extraction approach, and
develops, tests, and evaluates ways in which the method may be employed for design problem
reformulation.
1.3 Objectives

The objectives of the thesis are:

1. To establish a set of six behavior criteria for a design problem reformulation method through (a) a review of existing AI and machine learning in design research approaches, (b) a methodological analysis of findings in statistical natural language processing, and (c) a theoretical analysis of findings in cognitive neuroscience and cognitive science; To rationalize the choice of candidate algorithms and the approach for the method in relation to the behavior criteria (Chapters 2 and 3).

2. To present the computational method REIFORM for inductive acquisition of semantic design knowledge from the symbolic-mathematical syntax of design formulations (Chapter 4).

3. To apply REIFORM on problems from various engineering design domains, varying problem complexity (size and interaction/coupling), and different representational forms (analytical, non-analytical) for a range of design problem reformulation tasks, and to evaluate the performance of the method against other reformulation or solution approaches in terms of solution quality (Chapters 5, 6 and 7).

4. To present heuristics for the selection of values for user-controlled parameters in REIFORM (Chapter 8).

5. To present a set of methodological-theoretical postulates relating the semantics of design knowledge and its syntactic representation in design problem reformulation based on an assessment of the performance of the method (Chapter 9).

6. To present future possible extensions and applications of the REIFORM method (Chapter 10).

1.4 Research claims, contributions and significance

1.4.1 Design methodology

From a methodological perspective, the development of REIFORM was analogically inspired by applications of singular value decomposition, dimensionality reduction and similarity measurements in statistical natural language processing and digital image processing. It rests on viewing the design reformulation problem from an unsupervised pattern recognition and extraction perspective. Such a perspective offers several methodological advantages –

Knowledge-lean:
REIFORM is knowledge-lean and requires almost no design domain or task specific knowledge about the design problem to be pre-coded into the system. This is demonstrated in Chapter 4, which presents the full method description along with its application on an illustrative problem. The data representation step describes how an occurrence matrix is generated from a problem formulation example. REIFORM is applicable for any design problem that can be represented in the form of this standard occurrence matrix that captures a mapping between the syntactical symbols and expressions (mathematical or logical) used in the design experience. The data in the occurrence matrix is the only knowledge that is provided to REIFORM.

**Training-lean:**

REIFORM is training lean and can be used to inductively acquire and infer design knowledge over just one syntactic design model (as encoding of a semantic design experience) involving very few functional/interaction/dependency relationships (as encoding of semantic design episodes). REIFORM’s performance is also consistent when the number of episodes in a design experience is large. Further, the acquired knowledge can be used in the same design experience to reformulate the problem.

In design, the same problem can be formulated and solved in a number of different ways (Cagan et al., 1997; Ellman et al., 1998). It is common for design problems to have differing formulations, even if they belong to the same design domain. Two designers may choose to model the same problem in different ways. Often, the specific modeling assumptions and design requirements are never exactly the same, even if the same design problem is being modeled again. This results in different formulations. The problem formulation may be guided by the solution algorithm choice – the same problem could have very different formulations depending on whether a genetic algorithm or a gradient based optimizer is being used to solve the problem. In short, large training databases of similar symbolic-mathematical formulations may not be available to train an algorithm designed for acquiring or inferring problem modeling and reformulation knowledge. Efficient inductive knowledge acquisition and inference from just one design experience, and the use of this acquired knowledge in the same experience is, therefore, a useful property.

Chapters 4, 5, 6 and 7 show examples of design problems of varying complexity, and demonstrate in each case that REIFORM is able to use one design experience to acquire and infer design knowledge that is then used to reformulate the problem in the same design experience.

**Generality:**
REIFORM is general, and can be applied to problems from a variety of design domains, problem complexity (size, interaction/coupling) or mathematical representation forms (analytical, non-analytical). Chapters 4, 5, 6 and 7 present a range of problems to demonstrate this.

Further, the structure and behavior of REIFORM make it suitable for any problem reformulation task that could be performed by measuring and interpreting associative relationships between sets of symbols in design representations. The problems in Chapters 4, 5, 6 and 7 are chosen such that a range of reformulation tasks is demonstrated using this single algorithm, i.e. the same processing steps can be applied to perform different reformulation tasks. REIFORM exploits an underlying conceptual similarity in the different reformulation tasks, despite the different type and focus for each task – for each specific reformulation task it uncovers the implicit global structure of associations between events and episodes from the explicit local structure of the problem representation. The performance of REIFORM (computation load, solution quality) is compared with the performance of the different algorithms used in the source papers to solve these different problems.

Capturing multiplicity and invariance in design knowledge:
REIFORM can be used to infer multiple reformulation decisions from the same problem representation. It also has the potential to preserve the invariant aspects of design knowledge in a design representation. The problems in Chapters 5, 6 and 7 demonstrate this.

The existence and exploration of multiple patterns in an open set of possibilities instead of one or a defined set of “right answers” in a closed bounded set is a characteristic that differentiates the modeling phase in design from traditional AI problem solving. The activity of design often involves multiplicity in interpretation; at the same time, some aspects of the design problem may be perceived as invariant and stable. From an AI problem solving perspective, there is a state space of solutions and a search method that is defined before the search for a feasible or optimal solution within this space commences. Such a search phase is characterized by the design optimization paradigm after the mathematical model has been defined. In modeling, this state space definition can potentially change every time a designer reformulates the problem. The modeling phase is characterized by design optimization or general symbolic-mathematical design before the mathematical model has been defined and while it is still in development. The modeling and search phases are often cyclic and iterative. Therefore, along with symbolic and numeric search algorithms that ensure the calculation of exact, unambiguous and optimal solutions, it is useful to have exploratory tools and methods that can allow a designer to see the same problem in different ways (multiple interpretations) and provide insight into defining the state space of solutions before the search commences.
In summary, the main methodological advantages offered by REIFORM as a design problem modeling and reformulation support method are that it allows a designer to perform certain types of reformulation tasks in an exploratory manner, and can acquire and infer knowledge without the need for a large training database or high level knowledge engineering to do so. It is simple from an implementation perspective – it can be used with minimal amounts of programming effort, and allows for quick redefinitions of the problem and subsequent observation of changed reformulation results. The methodological contributions presented in this section map directly to the behavior criteria C1 to C6 established for REIFORM in Chapters 2 and 3 (Table 1-1). Chapters 4, 5 6 and 7 demonstrate that REIFORM is able to fulfill these criteria. Chapter 9 assesses the performance of REIFORM against these behavior criteria.

<table>
<thead>
<tr>
<th>CLAIM</th>
<th>CRITERION</th>
<th>CHAPTERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knowledge-lean</td>
<td>C1</td>
<td>2</td>
</tr>
<tr>
<td>Training-lean</td>
<td>C2</td>
<td></td>
</tr>
<tr>
<td>Generality</td>
<td>C3</td>
<td></td>
</tr>
<tr>
<td>Multiplicity and Invariance of design knowledge</td>
<td>C4, C5, C6</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 1-1: Summary of claims and criteria mapping

1.4.2 Design theory

The research uses findings from cognitive neuroscience and cognitive science to establish behavior criteria C4, C5 and C6 for REIFORM and evaluates whether it fulfills these. These performance criteria define abilities that are not readily explained by pure “symbolic AI” approaches and are not easily measured in a quantitative way by computational efficiency measurements alone. These include uncovering patterns of “implicit” knowledge that are not readily encoded as logical rules or stated explicitly in the design representation, and automating a range of reformulation tasks that require the associative transformation of sets of symbolic inputs in design experiences.

The thesis presents how the “symbolic AI” approach to design automation may be extended by incorporating another lower level “rule”. This rule has two parts: (a) global implicit knowledge exists between symbols and can be inferred from their local contextual co-occurrence patterns in design representations; and (b) the inference is made possible by using an unsupervised pattern recognition and extraction approach that leads to the acquisition of explicit and implicit semantic meaning from design representations.

In the symbolic AI “rule-based” paradigm (Newell & Simon, 1976), there are explicit representations of symbols, explicit mappings of a symbol to its semantic meaning and explicit symbolic operations relating these symbols to characterize inference processes. This has been the theoretical basis of developing knowledge based expert systems. However, the performance of REIFORM suggests that the seat of semantic design meaning may lie as much
in the global patterns of relationships between symbols, implicitly distributed across the whole system of symbolic representation, as it lies in the explicit local mapping of a symbol to its semantic meaning. The thesis, through an analysis of the performance of REIFORM, presents a set of theoretical postulates in Chapter 9 addressing an alternate perspective on how symbols may interact with each other in design experiences to reify semantic knowledge in design representations, and how this can form the basis of design computation tasks like problem reformulation.

1.5 Thesis structure

Chapter 2 establishes the first three behavior criteria for the method based on a review of existing approaches. It builds the methodological basis for the choice of SVD, dimensionality reduction, and unsupervised, pattern based similarity measurements as component algorithms for the design problem reformulation method REIFORM. It presents an analogy based argument for the structural and behavioral role played by these three component algorithms (SVD, dimensionality reduction and similarity measurements) in the statistical natural language processing and digital image processing domains. It concludes by presenting the role they could play for design problem reformulation tasks, and how they satisfy the computational criteria established at the beginning of the chapter.

Chapter 3 explains the methodological choice of algorithms in Chapter 2 by presenting a theoretical discussion on findings in cognitive neuroscience and cognitive science. This discussion on how human minds engage in semantic knowledge acquisition through symbol processing and resulting characteristics inherent in symbolic knowledge representational systems establishes three additional behavior criteria for REIFORM’s performance.

Chapter 4 describes the REIFORM method and demonstrates it step by step on an analytic design optimization problem.

Chapters 5, 6 and 7 present the application and evaluation of REIFORM for various problem reformulation tasks. Chapter 5 focuses on design decomposition and modularity analysis. Chapter 6 focuses on constraint satisfaction tasks like topology design and facility layout planning. Chapter 7 focuses on the identification of linked design variables and design “case” identification.

Chapter 8 establishes heuristics for choosing user controllable parameters such as similarity measurement threshold levels and dimensionality. The choice of these parameters plays an important role in (a) getting to the final problem reformulation decisions, and (b) using REIFORM for problem exploration and getting multiple reformulations and decisions. Studies and heuristics are presented that guide the selection of these parameters.
A summary assessment of the performance of REIFORM against the behavior criteria presented in Chapters 2 and 3 is presented in Chapter 9. The chapter then presents a set of theoretical postulates that result from an analysis of the performance of REIFORM. The postulates relate semantic design knowledge, its syntactic representation, computable reification relationships that exist between the two, and how this view can be useful in reformulating problems.

Chapter 10 presents a summary review of claims and conclusions and directions for future work.
Chapter 2

An Analogy Based Solution: Semantics from Syntax

So, “semantic” properties are connected to open-ended searches because, in an important sense, an object’s meaning is not localized within the object itself...Thus, another way of characterizing the difference between “syntactic” and “semantic” properties is that the syntactic ones reside unambiguously inside the object under consideration, whereas semantic properties depend on its relations with a potentially infinite class of other objects, and therefore are not completely localizable. There is nothing cryptic, or hidden, in principle, in syntactic properties, whereas hiddenness is of the essence in semantic properties.

Douglas R. Hofstadter, Godel, Escher, Bach

Design problem formulation and reformulation is concerned with the transformation of design semantics into a formal mathematical model. Design semantics are the “meaning” of a design as conceived by a designer. Often, the mathematical model is a result of the personal subjective choices exercised by the designer about objective information content, e.g. the physics of the system being modeled, the requirements specified by the client, etc. The compound of these choices made by the designer forms the final semantic meaning of the design work – the design elements and relationships that describe the structure and behavior of a design object as interpreted by the designer. The mathematical model reifies these design semantics as symbols and functions.

To classify the subset of tasks that come under problem reformulation, consider some of the major questions that designers face while creating a design representation (Ellman et al., 1998; Papalambros & Wilde, 2000): which structural or behavioral quantities to represent, which design elements to represent as variables, which ones to fix as parameters, what relationships between these variables and parameters to consider as objective functions or as constraints, how to decompose a large design problem, how to reformulate problems into
mathematically simpler forms so that they become easier to solve, etc. Once a design model is constructed, solution algorithms are applied to a defined problem space, and optimal or feasible solutions found. Further, a reformulation is often dependent on results observed from a prior formulation. Though reformulation and solution search are cyclic, iterative processes, how are design models reified in the first place? What is the relationship between a symbol and the meaning it encodes?

Designers encode the “meaning” of a design object in a symbolic representation. They construct such representations on the basis of experience-based knowledge, formal domain knowledge, mathematical modeling skills, and current design problem requirements. However, from an AI standpoint, there is no obvious “rule” based answer to the problem reformulation process – there is no well-developed computational process that takes in abstract modeling requirements as input and produces a symbolic mathematical representation as output. If cast as an AI / learning problem, this is an input – output mapping is of the following form: \( f(\text{abstract modeling requirements, semantics}) \rightarrow (\text{symbolic design model, syntax}) \). This mapping between semantics and syntax is not a simple one-to-one direct one but a complex, multi-faceted one. Trying to go from the abstract semantics to syntax is too hard an AI / learning problem to be tackled by known methods if cast in this form. Thus, the motivation of this research is to develop a computational method that assists with design problem reformulation tasks by acquiring the semantic structural-behavioral knowledge of the design from its syntactic representation and using the acquired knowledge to reformulate the same syntax. In other words, the mapping described above is recast as an implicit recursive relationship: \( f1(\text{initial syntax}) \rightarrow (\text{extracted semantics}); f2(\text{extracted semantics}) \rightarrow (\text{reformulated syntax}) \). If a syntactic design representation is taken to be an encoding of a design experience, how can we develop a computational method that acquires/infers semantic design knowledge from a formal representation, and uses this knowledge to reformulate this same representation?

2.1 Review of existing approaches: establishing the first three behavior criteria for the method

The application of AI and learning algorithms to design problems is a wide area of research. Extensive reviews (Duffy, 1997; Grecu & Brown, 1998; Sim & Duffy, 1998) show that most research endeavors focus on developing specific knowledge acquisition and inference methods for specific design domains, representational forms or for specific design tasks. This is understandable because the authors note that applying and developing AI techniques for design are difficult. Design problems require knowledge from various domains, use different kinds of representations, and design (as problem solving) involves multiple, simultaneous
reasoning abilities such as analysis, abstraction, evaluation and explanation (Grecu & Brown, 1998). These cognitive activities are difficult to model computationally as their cognitive mechanisms are largely unknown. “Stitching” them together to describe design computationally is orders of magnitude more difficult.

More specifically, much research has focused on combining artificial intelligence and machine learning techniques to improve upon and automate aspects of design optimization. Cagan et al. (1997) present a review of artificial intelligence techniques for optimization in engineering design. The review reinforces the observation that the explicit symbolic reasoning perspective has been a dominant one in AI applications developed for design computation. Schwabacher et al. (1998) explore supervised inductive machine learning techniques such as decision tree induction for automating various tasks in design optimization, including formulation selection and synthesis. Other notable research exploring similar issues include work by Ellman et al. (1998) and Gelsey et al. (1998). Campbell et al. (2003) present a design synthesis tool called A-Design, in which agents in a multi-agent system evolve conceptual design objects based on genetic algorithms, asynchronous teams and functional reasoning. Moss (2004b) explores learning in such a system, where useful “chunks” of design knowledge are learnt and used in future design tasks. All these approaches either require a high level of knowledge engineering (as rules, heuristics, grammars, functional/qualitative reasoning mechanisms, etc.) or they require a large training database of solved examples of various problems for the techniques to exhibit useful learning and inference characteristics.

From a pragmatic engineering standpoint, it will be useful to develop a method that is knowledge lean, training lean and widely applicable across design and representation domains for the following three reasons.

One, it is impractical to have a system with high levels of embedded knowledge engineering for automating symbolic design reformulation tasks. Choy and Agogino (1986) present an example where an established symbolic optimization method, monotonicity analysis (Papalambros & Wilde, 2000), is automated using a symbolic reasoning approach. This suggests that for methods that are complete, i.e. have a well-defined scope of application, it is possible to build a “complete” automated system. However, such complete methods may have inbuilt modeling assumptions that all design problems may not satisfy. Monotonicity analysis based methods, for example, require that the functions be differentiable and monotonic. Schwabacher (1996) presents techniques for reformulation selection and modeling constraints for learning “good” parts of the solution space in order to guide the numerical algorithm. However, in such approaches, the design engineer has to manually encode the rules specific to a design domain using a specific grammar. One significant point to note in these approaches is that a problem is reformulated based upon the information that is already available within a current formulation, either symbolically or numerically.
In terms of domain knowledge, for most design problems, designers do not know what the “best” formulation is in advance. Significant trial and error is involved before a formulation is finalized (Ellman et al., 1998). Thus, it is hard to have a standard knowledge base or a structure of explicitly stated “domain rules” that can be applied to design problems to produce formulations and reformulations. Secondly, designers frequently depend upon a large amount of background or domain knowledge to accept or reject formulation decisions (Gelsey et al., 1998). It may not be possible to have such knowledge encoded explicitly as rules, or it may require too large an effort to engineer a system that has such knowledge explicitly encoded.

While knowledge-driven strategies and techniques make a design computation system powerful, they also require much effort to build and maintain. Therefore, in addition to knowledge-rich approaches, it may be beneficial to explore knowledge-lean approaches. Therefore, as behavior criterion C1 for a computational method, it will be useful to have a method that does not require high levels of knowledge engineering.

Two, it is often difficult, if not impossible, to provide an automated system with a large database of training cases for learning design formulation. While it is possible to automatically generate a training database in numerical optimization cases (Schwabacher et al., 1998), generating a training database for symbolic cases will be more difficult. Given the specific conditions related to each problem, formulations from the same design domain or even the same problem in different settings can have widely differing mathematical forms (Ellman et al., 1998). Thus, it is difficult to define the learning characteristics and problem representation form for a training database in any general sense. However, the fact that human designers often use the information available in a current formulation to reformulate the problem suggests that one possible seat of learning or knowledge acquisition can be within the same design experience. Therefore, as behavior criterion C2 for a computational method, it will be useful to have a method that is training lean and can use the knowledge acquired in a design experience to act in the same design experience and reformulate the problem.

To cast this as a knowledge acquisition or inference problem, one could say that a single problem formulation is a training set of sorts. The human has given the system a set of explicit relations through symbols and functions. The role of the system will be to acquire/infer possible sets of semantic meaning implied by these explicit relations and then use those to reformulate the problem.

Three, it is expensive to develop, build and maintain design computation and automation systems and methods, and have them applicable only to specific design domains or tasks. Therefore, to augment the capacities of knowledge-rich specific methods that focus on optimal solutions (for example, systems in which AI methods and techniques are combined with numerical optimization as done by the cited examples), it may be beneficial to have exploratory modeling and reformulation methods that allow a designer to develop insight and
heuristically explore the problem for various kinds of reformulation tasks at the pre-optimization stage. For example, one learning problem in numerical optimization is to “seed” a good starting solution. Within the space defined by a given design model, if an algorithm learns the “good” regions, then it can reach the optimal solution more easily. If we cast the same problem one level up and in symbolic form, another learning problem is to “seed” a good model. If an algorithm, starting from a given initial design model, can infer “good” reformulations, then it may be tractable to compute an optimal solution since the “good” reformulation is well-behaved or bounded. Inherent in this idea is the notion of multiple interpretations. It is fairly well established that in design there is no “one” right formulation, and there can be many correct formulations. A heuristic exploratory method that provides the insight to infer multiple reformulations from a single formulation will be useful. Further, as design problems can be modeled using various representational forms (analytical, non-analytical) and can be of varying complexity (size of problem, degree of interaction between variables), it will be useful to have a common representational form that can operate over all types of design problems. As behavior criterion C3, it will be useful if such a method allows heuristic problem exploration by inferring multiple useful interpretations, and is applicable across problems of varying complexity from various design domains and stated using different representational forms.

C1, C2 and C3 establish the design methodology based criteria for the method.

2.2 Approach used in this thesis

In contrast to other methods based on high level knowledge-engineering or supervised learning, the theoretical and methodological approach presented in this thesis views the design reformulation problem from an unsupervised pattern recognition and extraction perspective. The method presented here focuses on the modeling and reformulation stage with the motivation of fulfilling the three design methodology based criteria identified in the previous section. One possible approach to address all these criteria together would be to focus on the knowledge available in the syntax of a single formulation. That is, the mathematical design model itself (from any domain using any representational form) is the basis for knowledge extraction with the assumption that it encodes a large part of semantic knowledge applied in a design experience. No extra knowledge is required to be encoded if the assumption is that the design model itself contains the knowledge needed for reformulation. No extra training cases are required if the assumption is that the set of explicit symbolic relations in a design model are the training set. Therefore, the knowledge acquisition/inference/learning problem is cast in general terms of acquiring semantic design knowledge from syntactic design representations. The research investigated other knowledge
domains where the learning problem is cast in an analogically similar form – acquiring semantics from syntax.

The approach presented in the thesis is inspired by an analogical transfer of ideas from the statistical natural language processing (SNLP) and digital image processing (DIP) domains. In particular, it is inspired by the role played by the linear algebra based factorization method Singular Value Decomposition (SVD) in various knowledge domains to address the specific problem of extracting semantic patterns from a syntactic representation. In SNLP, the Latent Semantic Analysis (LSA) method uses SVD (Landauer & Dumais, 1997) to reveal semantic patterns in textual data. The underlying idea in LSA is that linguistic knowledge contains a large number of weak correlations between semantic concepts, and that these latent correlations are captured in a distributed contextual manner by the syntax of language. The main claim is that SVD and the correct choice of dimensionality to view these relations are able to reveal the semantic patterns and the “latent” semantic meaning of the text. In digital image processing (Kalman, 1996; Strang, 2003), SVD is used as a mathematical tool to identify pattern redundancy in image data for compression of images. Further, this research found that SVD, as an algorithm, has the capacity to be used as a general pattern extraction mechanism in vastly different domains. Some examples (not exhaustive) include language (Landauer & Dumais, 1997), design text, content and team performance analysis (Dong, 2005), prediction of psychological phenomenon (Wolfe & Goldman, 2003), digital image processing (Kalman, 1996), internet search algorithms (Strang, 2003), and clustering gene microarray data (Liu et al., 2003). In all these approaches, the underlying “common” representation on which SVD operates is a matrix that captures relationships between one or two types of “things” or “concepts”, commonly one type syntactically defining and occurring in the other type. SVD was found to be used in conjunction with or for dimensionality reduction, similarity measurement and clustering techniques in all these domains. While dimensionality reduction, similarity measurements and clustering are commonly adopted approaches in machine learning and AI algorithms (Duda et al., 2000; Russell & Norvig, 2003), it is the role of SVD that is mathematically intriguing and interesting. Despite being used in such diverse domains for pattern extraction, it is not a commonly discussed method in an AI or machine learning textbook.

This research was intuitively inspired to explore this issue, the intuition deriving from a general observation – the semantic concepts of any knowledge domain are interpretively contained in the syntax of representation. Though part of semantic meaning is symbolized explicitly and locally through syntax, it is also generated implicitly and globally through the same syntax. Therefore, semantic meaning is not designated by a symbol and fixed; it is also a result of the interaction of symbols as determined by the syntax. If the local context of other
symbols in which one symbol appears changes, then the meaning of the symbol can change because the interaction between symbols has changed.

Consider what happens in the language and image processing domains. The empirical patterns of occurrence (words in sentences) capture inherent “meaning” in natural language; redundancies of pixel occurrences in an image capture the inherent graphic patterns of “meaning” in images in terms of graphic objects. The representation captures how symbols occur together in experiences. SVD manages to bring out implicit (“latent” in the LSA terminology) or explicit empirical syntactic patterns of occurrences of events or occurrences in their contextual episodes where the co-occurrence of the events themselves defines these episodes.

Engineering design, as a discipline, is vastly different from all these domains. In design optimization, mathematics, with its precise syntax, is the main representational mechanism. Mathematics, as a formal language, has characteristics very different from natural language. It has almost no ambiguity and very precise syntactic-semantic relationships. It is the primary symbolic language employed by designers for constructing representations in design optimization. Still, it is a “language” where symbols come together to produce meaningful symbolic systems of representation, in this case, the representation of designs. Based on an analogy drawn on structural (data representation) and behavioral (method performance) levels, this thesis conjectures that an application of the SVD, combined with dimensionality reductions and unsupervised similarity measurements to mathematical design optimization models, could nonetheless reveal an interesting parallel observation – that SVD should be able to reveal semantic design patterns from the syntactic, symbolic or mathematical representation of a design. The structural and behavioral analogies are presented in detail in the next sections of this chapter.

The thesis, therefore, proposes the following hypothesis – the mathematical-symbolic representation of a design captures the structural-behavioral characteristics of the design work being modeled, as semantically conceived by the designer. The variables and parameters represent structure and behavior concepts, and occur in functions or interaction/dependency mappings with each other. Thus, behavioral relationships are captured through the syntactic structure of the formulation. Changes in this representation as the optimization process passes through formulations and reformulations will reflect how the modeling of the engineered object changes, as well as how the designer changes his / her choices and decisions on the modeling.

There is some support for this hypothesis from research applying natural language processing to design. Dong and Agogino (1997) used computational linguistics based methods to explore the construction of design representations from textual documentation. Dong (2005) developed the Latent Semantic Approach for document analysis to show that
designers develop a “shared understanding” of a design object that is inherently captured in textual design documentation. Moss et al. (2004a), through empirical studies on differences in expert and novice behavior in designers, proposed that internal and external knowledge representation mechanisms and changes to them capture the structure and content of a domain and internal cognitive process content. In these studies, the implicit assumption is that the textual-syntactic representation of a design captures both the semantic choices exercised by designers as well as objective knowledge about the design object itself and that the knowledge is distributed throughout the syntax rather than being explicitly stated in a particular clause. These provide evidentiary support for the hypothesis – Since SVD is a domain knowledge independent algorithm operating only on the syntax, when “words” are analogically/conceptually replaced by “variables and parameters” and “sentences” by “functions, objectives, and constraints or interaction/dependency mappings”, then the same idea should remain valid – the symbolic mathematical representation of a design should capture both the semantic choices exercised by designers as well as objective knowledge about the design itself.

2.3 Analogies

To build a method for design reformulation, this section presents a detailed discussion on the analogy identified from Latent Semantic Analysis (LSA) (Landauer & Dumais, 1997) in statistical natural language processing (SNLP) and image compression methods (Kalman, 1996; Strang, 2003) in digital image processing (DIP). Both involve the use of the linear algebra based factorization method of Singular Value Decomposition (SVD). In SNLP, SVD-based LSA is used to reveal semantic patterns in textual data based upon distributed textual co-occurrences of words rather than individual word meaning. In DIP, SVD is used to identify pattern redundancy in image data for compression of images. SVD, applied in these two diverse domains, suggests a connection between semantic knowledge and its syntactic representation. To review the hypothesis presented in the previous section, a symbolic-mathematical design representation is the description of semantic design knowledge, i.e. “structural or behavioral elements-in-behavioral relationships” as “symbols-in-functions” in a syntactic representation. The conjecture is that SVD will be able to reveal the connections between the syntax of design representation and design semantics.
2.3.1 Structural analogy

2.3.1.1 LSA, analytic design formulations and non-analytic incidence matrix design formulations

In LSA, a corpus of linguistic data is converted into a word-by-document matrix. Rows represent words and columns represent documents in which words appear (Figure 2.1). The matrix entries are a measure of the number of times a word appears in a specific document.

<table>
<thead>
<tr>
<th>A</th>
<th>d1</th>
<th>d2</th>
<th>d3</th>
<th>d4</th>
<th>d5</th>
<th>m1</th>
<th>m2</th>
<th>m3</th>
<th>m4</th>
</tr>
</thead>
<tbody>
<tr>
<td>human</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>interface</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>computer</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>user</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>system</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>response</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>time</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EPS</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>survey</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>trees</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>graph</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>minors</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 2.1: LSA word-by-document example from (Landauer & Dumais, 1997)

Drawing a structural analogy between words-in-sentences (natural language) and design variables-in-functions (analytic mathematical language), a similar matrix representation could be developed for analytically stated design problems. Rows would represent design events (variables, parameters) and columns would represent episodic relationships (objective functions, constraints). The matrix could thus represent the semantic “meaning” of the design work as the relations between variables and functions. Structure and behavior variables can be combined and recombined to achieve different behaviors and functions, just as words can be infinitely recombined to achieve different texts and meanings. Figure 2.2 shows an example formulation – an analytical, non-linear, single objective optimization model for a hydraulic cylinder problem. Figure 2.3 shows the matrix representation for this problem. Each entry indicates whether or not a variable / parameter occurs in a function. This called the occurrence matrix.

Figure 2.2: Single objective hydraulic cylinder design problem (Papalambros & Wilde, 2000)
In the LSA terminology (Landauer & Dumais, 1997) the matrix captures how events (words) occur in episodes (sentences). Thus, in the analogy, each variable or parameter is an event, that occurs in an episode that is a function. This matrix form plays a crucial role in embedding the multiple pathways by which relations between events and episodes exist. It is these pathways that SVD uncovers. Mathematics as a formal language for design representation has the advantage of defining precise relations between variables and functions. Yet, it also has the disadvantage of the formality of mathematics in that the relation between input and output, variable and function, is largely fixed; sophisticated graph-based techniques are needed to “unfix” or “reveal” the relations (Michelena & Papalambros, 1997; Wagner & Papalambros, 1993a, b). Parallel to the language based approach, it will be useful if a knowledge-lean computational method is able to reveal how multiple implicit meanings can be contained within a single representation.

The occurrence matrix form can be used to represent both analytically and non-analytically formulated problems. The design incidence matrix form (Li & Li, 2005) or the Functional Dependence Table (FDT) form (Michelena & Papalambros, 1997; Wagner & Papalambros, 1993a, b) is an established method for representing an analytic formulation as a non-analytic matrix formulation, or for representing dependency, interaction or coupling relationships, in general, between design variables and functions or attributes or computational procedures. It has a direct structural correspondence with the word-by-document matrix in LSA.

In this general structural analogy, the matrix indicates how elements occur in the local context of their appearance. A notable difference is that linguistic analysis uses very large matrices derived from very large corpora of natural language. In contrast, matrices from the design domain are restricted to the size of the individual design problems and are likely to be much smaller in size. It is not a priori obvious whether SVD could uncover the multiple pathway relations between variables and functions, and this is an important research question.
2.3.1.2 DIP and non-analytic Design Structure Matrix (DSM) formulation

In the DIP domain, SVD is widely used for image compression tasks. An image is converted into a matrix of \( m \) rows and \( n \) columns; the \( mn \) matrix entries are a measurement of pixel values ((Kalman, 1996), Figure 2.4).

A structural analogy between pixel-to-pixel mappings (image processing) and design element-to-design element mapping (non-analytic design representation) can be drawn. A similar matrix representation already exists as an established non-analytic form of design problem representation – the Design Structure Matrix (DSM). The example DSM representation (Figure 2.5) shows an automotive problem with 16 design components (Pimmler & Eppinger, 1994) and their interactions. The analogy, at this point, is only drawn in a structural way, as no obvious similarity exists between what the image matrix represents and what the DSM represents. In a general structural sense, the matrices correspond to relationships that exist (or not) between the same elements in rows as well as columns. While image matrices can be square or rectangular, DSM matrices are always square as they capture mappings between the same set of design elements represented in the rows as well as the columns. This makes the DSM representation slightly different in form to the incidence matrix / FDT form where mappings are captured between variables and functions.

Figure 2.4: a 24 × 24 image (Kalman, 1996)
2.3.2 A common representation form from the structural analogy

In this thesis, all problem forms, whether initially stated in analytic or non-analytic form, will be converted into a common representational form, the occurrence matrix \( A \). The matrix \( A \) can be rectangular or square, depending upon whether it derives from an analytic formulation, an incidence matrix / FDT form, or a DSM form. The first two will generally produce rectangular matrices, while the DSM will produce square matrices. The occurrence matrix captures a common mapping how design variables, parameters or components come together in functions, design interactions or dependencies in the design problem representation. The mapping is equivalent for an analytical or non-analytical formulation.

2.4 Behavioral analogy

However diverse the structural analogies may appear, they seemed deserving of a deeper analysis. The research, therefore, turned to a behavioral analysis of the mathematical structure of SVD to draw a behavioral analogy.
2.4.1 What does SVD do?

SVD takes a general rectangular matrix $A$ with $m$ rows and $n$ columns and decomposes it into a product of three matrices, $A = USV^T$, where $U(m \times m)$ and $V^T(n \times n)$ are the left and right orthogonal matrices and $S(m \times n)$ is a rectangular matrix with non-negative singular values on the diagonal in order of decreasing magnitude. The number of singular values is $r$, where $r$ is the rank of $A$. The mathematical idea (Figure 2.6; (Strang, 1993, 2003)) is as follows: the row space of $A$ is $r$-dimensional and inside $\mathbb{R}^n$, and the column space of $A$ is $r$-dimensional and inside $\mathbb{R}^m$. We choose special orthonormal bases $V = (v_1, v_2, \ldots, v_r)$ for the row space, and $U = (u_1, u_2, \ldots, u_r)$ for the column space, such that $Av_i$ is in the direction of $u_i$. $s_i$ provides the scaling factor and $A v_i = s_i u_i$. In matrix form, this becomes $AV = US$ or $A = USV^T$. $A$ is thus the linear transformation that carries orthonormal basis $v_i$ from space $\mathbb{R}^n$ to orthonormal basis $u_i$ in space $\mathbb{R}^m$. Neglecting the null spaces, $A(m \times n) = U(m \times r) \cdot S(r \times r) \cdot V^T(r \times n)$. Another way to say the same thing is that a general rectangular matrix $A$ has been diagonalized into two independent spaces described by orthogonal bases $U$ and $V$, and related to each other by the magnitudes of the singular values. Three points relevant for analysis are:

1. Any rectangular matrix that captures a mapping between two types of “things” or “concepts” in any knowledge domain can be subjected to the SVD decomposition. The effect this has is that the dependent data in the original matrix is re-represented as a set of independent concept vectors in terms of orthonormal bases. These independent orthonormal vectors can now be linearly combined to produce the original data or approximations of it.

2. To perform a dimensionality reduction over matrix $A$, keep the first $k$ values of $S$ and produce a $k$-reduced truncated approximation of $A$ as $A'(m \times n) = U'(m \times k) \cdot S'(k \times k) \cdot V'^T(k \times n)$. Different approximations will be produced for different $k$ values. The entries in $A$ will change to lower or higher values in $A'$, producing a linear least squares approximation of matrix $A$. Now $U'$ and $V'^T$ imply that each row or column element type, instead of being described with $r$ components (of the left and right singular vectors) in an $r$-dimensional space, is being described in terms of a lower number of $k$ components in a $k$-dimensional space. The singular values preserve the most important associative relationships of the matrix $A$ in decreasing order of magnitude. Thus, a dimensionality reduction operation implies that the first $k$ singular values capture the most important patterns while those that are not retained in the approximation are “noise”.

3. SVD and dimensionality reduction operations convert the data in the matrix $A$ into a continuous, distance based representation. Each row or column element type from the original matrix is now a vector in $r$ (SVD) or $k$ (dimensionally reduced) dimensional
space as a combination of orthonormal vectors in $U$ and $V$. Any two can be compared to each other independently. The distance between these vectors is a measure of “similarity” between what the vectors represent. The predominant meaning of “similar” in unsupervised machine learning literature generally refers to conceptual similarity between two elements or objects. However, in this research, “similarity” implies that two events or episodes are similar if they are tightly bound in some way (through referential associations arising out of mutual co-occurrences in the original matrix). An obvious interpretation is that two points that lie close together are “similar”, and two points that lie far apart are “not similar”. For this work, to perform an unsupervised similarity measurement, a cosine distance is chosen, primarily because the cosine distance captures the magnitude as well as the direction aspects between two points in space.

These points will be relevant to the following discussion.

Figure 2.6: Orthonormal bases that diagonalize $A$ (Strang, 1993): $A$ is the linear transformation that carries orthonormal basis $v_i$ from space $R^n$ to orthonormal basis $u_i$ in space $R^m$

2.4.2 What does SVD do in LSA?

A general word-by-document matrix $A$ captures the local co-occurrences of words in documents. Each row represents how a word occurs across all documents and each column represents what words appear in each document. SVD of this matrix $A$ produces factors $U$ and $V$. The special mathematical behavior of SVD is that $U$ and $V$ contain orthonormal vectors, i.e. independent components derived from mutual co-occurrence information contained in the original data matrix. These independent components measure combinations / correlations between the elements in the original occurrence matrix, but are themselves uncorrelated to each other. In LSA, $U$ represents a conceptual “word” space and $V$ represents a conceptual “document” space. Singular values scale, i.e. stretch or contract the orthonormal vectors.
The next step in LSA is a dimensionality reduction step. In this step, a $k$-reduced approximation of the original matrix is produced. In behavioral terms, this implies that the original data is now re-represented (by the SVD) and observed in a reduced number of dimensions (by dimensionality reduction). The $k$-reduced linear least squares approximation is a best guess on whether the word $i$ appeared (or did not appear) in document $j$ in the original matrix. Preserving the largest singular values will capture the most important associations and ignore the weaker ones. In the LSA interpretation, the aim is not to re-create the original matrix perfectly by using a reduced rank. The principal claim in LSA is that at some optimal dimensionality, the $k$-approximation will cut out the “noise” or irrelevant relationships and will induce important implicit or latent relationships that exist between the words and documents and cannot be observed directly from the original matrix. For a demonstration, refer to Landauer (1997).

The conceptual explanation for this mathematical process is that linguistic knowledge contains a large number of weak interrelations. Although words are invariant, the way in which they come together in sentences is unique. Thus, the original data matrix is sparse, and this does not adequately capture implicit “meaning”. The implicit relations are determined by the SVD and dimensionality reduction process because of the capacity to employ the linear combination of each matrix entry with all the others in producing the $U$ and the $V$ and then observing this in a reduced number of dimensions. SVD creates a “global” space from “local” occurrences of words in documents, where the association strengths of words and documents with each other is a statistical measure of how each one relates to all the others, whether or not they appeared together explicitly in the data set. The idea is that it has to try to model the original meaning intent captured by the original document using a combination of linearly independent vectors.

For example, this computation would capture the semantic (global / implicit) relationship that the concept (word) “SVD” shares with the concept “linear algebra”, even though it may not appear directly in the title of a book “Introduction to Linear Algebra” (local / explicit relationship) because somewhere, in the whole data set, the words “SVD”, “linear” and “algebra” will occur with each other or other common words in a distributed way (implicit relationship).

The next step is to measure the degree or intensity of word-word, sentence-sentence or word-sentence associative similarity. Since the re-represented dimensionally reduced space already recasts the dependent data as independent points in continuous space, this similarity measurement is done in an unsupervised way. Given a query word, the aim is to retrieve all other words and documents that are semantically similar to the query word. Because linear combinations of original matrix entries were used to create the $U$ and $V$ spaces, all words and documents that statistically occur together more times will lie close together in the scaled $US$.
and SVT spaces (and equivalently those that do not will lie far apart). That is, distance is a measure of similarity, if similarity is measured by the strength of mutual associations between elements/objects. Therefore, in the dimensionally reduced re-representation space, words and sentences are vectors that can be assessed for similarity in terms of some distance metric. Usually, similarity is assessed between them through cosine distance measurements. Figure 2.7 shows an example from (Berry & Dumais, 1994) showing how words and documents are represented in a $k=2$ space and a query in this space that retrieves the words and documents within a certain cosine threshold. A higher cosine means higher similarity and vice versa.

2.4.3 What does SVD do in DIP?

In DIP, the focus is on producing a good approximation of the original matrix $A$ because the objective is data compression. The matrix $A(m \times n)$ is a representation of pixel values. SVD is performed on matrix $A$. A reduced rank approximation is found that best represents the original data. The rank $r$ of a matrix is a measure of the number of independent columns or rows of the matrix. Thus, it is a measure of the redundancy in the data, because the lower the rank, the higher the number of dependent rows or columns. This has a direct behavioral analogy with images. Any large scale feature in the image will tend to show up as redundancy in the matrix, as rows and columns will contain similar repeated values to represent this feature. This implies that some approximation of the original matrix will be able to represent, without any loss, the original data. The objective in data compression, therefore, is to find the best dimension that is able to reproduce the original data to a good degree of approximation. Note the difference in interpretation from LSA – in DIP error reduction and a lossless approximation is the aim.
2.5 Performance characteristics from SVD, dimensionality reduction and similarity measurement

Based on the above discussion, this section presents important behavioral characteristics of SVD, dimensionality reduction and unsupervised similarity measurement that are interesting for acquiring / inferring patterns from design representations.

2.5.1 Measure of “semantic similarity” in design representations

In a design representation, variables (or parameters, system components etc.) occur in functional or general interaction/dependency relationships with each other. In a design representation, if two variables $x$ and $y$ occur in many constraints together, this says that there is a strong “global” interaction between $x$ and $y$. On the other hand, if $x$ and $y$ occur in just one constraint together, then the interaction between them is not as strong outside the “local” context of that function. Similarly, if two constraints are defined by the same or many common variables, they are more “similar” than two constraints that are defined by different sets of variables. As a third comparison, if a single variable occurs in many constraints, then
this shows that this variable shares a strong relationship with all aspects of the formulation. This is the notion of “semantic similarity” or closeness used in this work – sets of events that tend to appear together in episodes within an experience have more associative connections with each other than sets of events that do not tend to co-occur in the same episodes within an experience.

One characteristic of SVD and dimensionality reduction approach is that the initial data in the matrix is converted into a continuous, distance based representation. The approach shows that it can transform the local patterns of associations between matrix entries into a representation that brings together in space “things” that contextually occur together more number of times, and equivalently pushes far into space “things” that do not occur together many times. This means that the design dependencies can be captured by a distance metric like a cosine measurement. That is, in the space produced by SVD, distance is a measure of the strength of relationship between two variables or functions. Therefore, in an unsupervised way, purely due to the structure and behavior of SVD, variables and functions that share design relationships will be pulled together in space. The formal development of this argument will be presented in Chapter 4.

This is a useful characteristic for design reformulation, as a primary concern in design reformulation tasks is to identify which variables and functions are linked together and to what degree. For the purpose of this thesis, the notion of “similarity” is defined as follows: Event $i$ and event $j$ (variables, parameters, design components) are similar if they occur together in the same episodes (functions, design dependency relationships) or if they occur, together or individually, with any other common events. Episode $i$ and episode $j$ (functions, design dependency relationships) are similar if they contain the same or overlapping sets of common events (variables, parameters, design components).

2.5.2 Dimensionality reduction and implicit pattern extraction

Design knowledge contains a large number of strong and weak semantic interrelationships. While constructing a design representation, designers choose to model some of these explicitly, while some of them are left latent, i.e. they are not explicitly represented. Consider an over simplified example: the concepts of area $a$, volume $v$, length $l$, breadth $b$ and height $h$. In a representation, two functions could be $a=lb$ and $v=lbh$. However, that volume is also area times height is an implied or latent relationship that exists in the semantic space but not in the representation, as there is no explicit representation for $v=ah$. More generally, each functional representation is one possible ordering or capture of a behavior through the symbols which occur in the function. Other behaviors may appear such that we decide not to include them in the explicit representation. In acquiring the semantic meaning of a design work from its syntactic representation, a computational method will need to extract both the
explicit as well as the implicit meanings. This is because a key requirement of reformulating a design problem is to generate the ability to “see” other possibilities and relationships that are not explicitly evident. A reformulation can rest on detecting and making implicit relationships explicit.

A hypothesis, based on the structural and behavioral analogy, is that the process of projection of symbols (U space) onto functions (V space), and vice versa done with SVD, followed by dimensionality reduction will locate these (multiple), latent relations. SVD and dimensionality reduction recreate the semantic meaning intended by the original combination of the symbol space and the function space as a k-level linear combination of the orthonormal bases. In doing so, since the first k singular vectors, say $k = 2$ to $a$, capture the most important associative patterns, and produce the “best” or the optimal least squares approximation to the original matrix, latent, implied relationships should be retrieved. Therefore, the dimensionality reduction step reveals patterns contained in the syntax that are not directly observable in the original occurrence matrix. This, if demonstrated, will be an important characteristic for problem modeling. Almost always, design representations are sparse. Symbolic design representations do not explicitly code all the semantic relationships, by necessity, choice or error. If the algorithm is able to extract implicit patterns in the natural language domain, then it should be able to do so for design representations.

### 2.5.3 Redundancy, matrix compression and explicit pattern extraction

The DIP discussion shows that if the data matrix contains redundancy, then, at some reduced approximation, the exact original data is reproduced, i.e. a lossless compression is obtained. This implies that for a large design problem, if the matrix contains some amount of redundancy, then this is an indication that the explicit relationships in the problem representation can be safely inferred at a lower approximation than the original, i.e. at some lower $k$ values.

Combining this feature with the LSA discussion, we can observe that the first few singular values, say $k = 2$ to $a$, will be responsible for capturing the implicit relationships. Then at some $a$, the approximation begins to return the same information as the original matrix, as the $k$ value approaches $r$. Therefore, the $k = a$ to $r$ (recall that $r$ is the rank of the matrix $A$) approximations will only return the original explicit information contained in matrix $A$. This type of ability limits the search for implicit meaning within a range of $k$ (2 to $a$) values and also provides a measure of how such implicit meaning is changing over the dimensions. In symbolic-mathematical design problem reformulation, we need the method to find the invariant relationships that will never change because they are represented locally/explicitly in the original problem formulation and therefore in the data matrix, as well
as the global/implicit, multiple relationships that are induced using the dimensionality reduction step.

2.5.4 Multiple and invariant patterns from a single data set:

Sections 2.5.2 and 2.5.3 showed that SVD and dimensionality reduction might be able to capture both explicit and implicit design relationships from a design formulation. The explicit relationships are important, because they show the invariant features of the original formulation. The implicit relationships are important because they show the possible relationships between variables and their relationships that are not explicitly available in the design formulation and arise from local association information in the original formulation. In an intuitive way, this is the seat of design problem reformulation – observing these implicit relationships will be similar to varying the “modeling freedom” because these implicit relationships may suggest multiple reformulation possibilities. Some of these, when made explicit, will change the problem formulation.

One characteristic deriving from the structural or behavioral analogies is that different measures or degrees of similarity will reveal different semantic groups from the same data set. There are two ways of extracting multiple patterns from a single data set: (1) deciding upon a $k$-value in the dimension reduction step (used in the SNLP and DIP domains to identify different approximations), and (2) deciding upon different cosine thresholds when using cosine distance as a similarity measure in the unsupervised similarity measurement step (used in the SNLP domain to measure semantic similarity between words and documents). Section 2.5.1 showed how the notion of semantic similarity is defined for design representations. For the same $k$, if the cosine threshold used to measure similarity is varied, then different similarity patterns may emerge. For different $k$ values, i.e. different dimensions, different similarity patterns may emerge. If, in the case of design representations, the patterns show a transparent relationship across cosine threshold values and $k$-values, and not some arbitrary, random behavior, then this is a useful property for design problem reformulation. In some design problems, there are multiple potential formulations that can be considered to be “good” formulations, and no dominant formulation. If this property holds for design representations, by varying the cosine thresholds and the number of dimensions $k$, different reformulations can be observed, because both these steps will retrieve the implicit relationships. This property makes the same training data set, i.e. the same problem formulation, capable of encoding multiple potential patterns for reformulations.
2.6 Summary

In this chapter, a review of existing approaches established three design methodology based behavior criteria, $C_1$, $C_2$ and $C_3$, for a design problem reformulation method. An analysis of the mathematical principles of SVD combined with dimensionality reduction and unsupervised similarity measurements was demonstrated as a possible approach for developing this symbolic design problem reformulation method. Structural and behavioral analogies with the role of the component algorithms in LSA and DIP were presented, and the main behavioral characteristics of the algorithms that seemed relevant to design were discussed. The analysis suggests that SVD can be used to extract empirical patterns of syntactic co-occurrences of variables and parameters in objective and constraint functions. By observing and inferring patterns imputed from different dimensional reductions of the original data and different similarity measurement or clustering thresholds, it was conjectured that the general approach outlined may be able to acquire implicit and explicit semantic association patterns from a syntactic representation. This knowledge may then be used for problem reformulation.
Chapter 3
Theoretical Conjectures: Why Does it Work?

I do not refer to the mathematical difficulties, which eventually are always trivial, but rather to the conceptual difficulties.

Science and the human temperament, Erwin Schrödinger

A review of existing approaches in Chapter 2 helped establish three design methodology based behavior criteria for the method. The chapter established the choice of candidate algorithms (SVD + dimensionality reduction + unsupervised similarity measurement) for a computational method to acquire semantics from design syntax using an unsupervised pattern recognition and extraction perspective. This choice derived from purely methodological considerations. Structural and behavioral analogies with the language and image processing domains were presented, along with the observation that SVD plays a successful role for pattern recognition and extraction in many diverse knowledge domains.

However, the choice of the algorithms still seems arbitrary, more to the effect of “looks like it will work, so let’s try it”. This chapter presents a deeper analysis based on human neuro-cognitive processes to argue why this mathematical mechanism seems to be an appropriate choice of method to uncover semantically useful patterns. Analyses of empirical and theoretical findings from cognitive neuroscience and cognitive science on how the human mind engages in symbolic forms of representation and communication are presented. Based on the cognitively-based analysis, some additional behavior criteria expected from the method are presented.

The thesis makes no claim to present a computational model for any cognitive phenomena. However, symbolic design problem formulation is a task which human designers perform well. Therefore, it may be useful to explore how the human mind engages in symbol processing and the resulting connections between semantics and syntax in symbolic systems of representation. Such an understanding provides additional metaphors and behavior criteria.
for the development of, specifically, the computational method presented here, and more generally, design theory issues on the role played by symbols in encoding local and global design meaning in symbolic-mathematical design representations.

3.1 What is the connection between design syntax and semantics?

In the past few decades, the symbolic AI view has guided the engineering approach adopted towards developing machine learning and AI algorithms. The world view that lies at the basis of this approach is the physical symbol systems hypothesis, which claims that “a physical symbol system has the necessary and sufficient means for general intelligent action” (Newell & Simon, 1976, pp. 116). This hypothesis claims that any system (human or machine) exhibiting intelligence must operate by manipulating data structures composed of symbols. Not only does this imply that the processes underlying knowledge acquisition, reasoning, learning and inference are symbolic, but also, more radically, the physical existence of knowledge in the mind is symbolic. In the decades that followed, there ensued a great debate concerning the truth or falsification of this hypothesis (Russell & Norvig, 2003). The main focus of this debate has been on the question of whether knowledge (in the mind) is symbolic (or sub-symbolic as claimed by the neural networks/ connectionist approach).

Whether knowledge in the mind exists in symbolic form (Simon, 1995), or is only represented symbolically (Clancey, 1997, 1999) is a lasting debate relevant to any discipline that concerns itself with intelligence and its artificial construction. Design is no exception. In design, this debate becomes particularly interesting for the symbolic-mathematical mode of designing. Whereas in visual, graphical or linguistic modes of designing, the representation itself is ambiguous and the encoding of ambiguity is considered a robust mechanism for accommodating multiple active ideas (Schon & Wiggins, 1992; Suwa et al., 2000), the mathematical modeling of designed objects does not lend itself to the same kind of ambiguity. In this mode of designing, the construction of a representation is a precise, well-structured, unambiguous encoding of knowledge that is otherwise inherently ill-structured, ill-defined and ambiguous.

Whatever be the form in which knowledge exists in the brain, we know that an observable result of the cognitive activity of designing is the symbolic-mathematical problem model. The following hypothesis generalizes the methodological hypothesis presented in the last chapter: 

If the symbol is a syntactic abstraction produced by the mind, then there is a relationship between the symbolic system of representation and the semantic content that it intends to represents. For design, this implies that there is a connection between the symbolic-mathematical representation and the semantic “meaning” of a design work as encoded in the representation.
3.2 Physical Symbol System Hypothesis: Symbols are [semantic] knowledge

The basis of symbolic AI, the physical symbol system (PSS) hypothesis, gives us the following understanding: “A PSS is simply a system capable of storing symbols (patterns with denotations), and inputting, outputting, organizing and reorganizing such symbols and symbol structures, comparing them for identity or difference, and acting conditionally on the outcomes of the tests of identity. Digital computers are demonstrably PSSs, and a solid body of evidence has accumulated that brains are also. The physical materials of which PSSs are made, and the physical laws governing these materials are irrelevant as long as they support symbolic storage and rapid execution of the symbolic processes mentioned above…” (Simon, 1995, p. 104).

Following this kind of symbolic AI logic, and referring to the example formulation presented in Chapter 2 (Figure 2.2), the symbol “i” is a reference to a physical quantity in the world being abstracted as “internal diameter for a hydraulic cylinder” rather than, say, the number of atoms required to fill a given volume of space. Similarly, symbol “s” is a reference to a physical quantity “hoop stress in the cylinder”. Explicit symbolic relations and operations/processes exist between these symbols. Simon’s preceding quote would suggest that there is no difference in the “i” as it exists in a computer program and the way “i” exists in the mind. Symbolic operations between “i” and “s” would be the same as inside a human mind and in a computer, and this is necessary as well as sufficient to explain all related knowledge and actions. Because the mapping is defined by designers, in such “rule” representations, there seems to be an explicit one-to-one mapping between a symbol and its meaning (the signifier – signified relationship). This interpretation is confirmed by this quote from the original Turing Award Lecture (Newell & Simon, 1976): “Designation. An expression designates an object if, given the expression, the system can either affect the object itself or behave in ways dependent on the object.”

However, there must be a difference between what symbols mean to a human being and what they mean to a computer. This is based on the evidence that neither human designers nor the symbol system hypothesis can completely or exhaustively explain the basis of why a designer chooses the “i” and the “s” and relationships between them in the first place – humans can produce design formulations and reformulations, but cannot always explain the explicit symbolic “rules” by which they do it (although it may be possible to do so retrospectively). A pure physical symbol system (most symbolic AI algorithms are exemplars based on the PSS theory), on the other hand, cannot perform this behavior very well or at all, else it would have been a routinely automated one.

From a methodological perspective, symbolic AI or expert systems based approaches take a “symbol” to “object in the world” relationship as a given. From this basis, a symbol
represents some object or construction in the world. Symbolic operations between these symbols are then encoded as “rules”, and this becomes the definition of “knowledge”. However, consider that in the natural language domain, the idea of a direct “symbol” to “object in the world” relationship (word-meaning, signifier-signified relationship) or the known “rules” of grammar cannot explain why language encodes “implicit” or “latent” meaning. The LSA method in SNLP, which purely measures syntactical associative patterns of how symbols co-occur with each other, is able to reveal implicit semantics that are not explicitly coded into the syntax. This suggests that semantic meaning is a higher-order phenomenon than what is directly or explicitly observable from the sparseness of its representation – the explicit one-to-one mapping between the symbol and its intended meaning is not sufficient to capture the complete semantic meaning encoded by the symbol.

3.3 Insights from cognitive science and cognitive neuroscience

Evidently, there is no one-to-one connection between a symbol and its meaning inside human minds, as is in a computer, whether that symbol is a linguistic (natural language) or a mathematical (design) representation.

3.3.1 Situated cognition: Perceptual-conceptual couplings re-coordinate to produce a symbol

Clancey (Clancey, 1997, 1999)) explains this dilemma by analyzing that the symbol does not mean the same thing in a computer and in a human. In a computer, it is an entity referring to a “thing” in an isolated, atomic way. In fact, it needs a human being to make even this interpretive association between the symbol and the thing it represents. Inside a computer, it is just symbols and associations between symbols – a “flat” relationship. Using the theory of situated cognition and constructive memory, Clancey proposes that, in human reasoning, a symbol is the result of dynamic relations, couplings between perceptual and conceptual categorizations. Such a categorization allows itself to be “re-structured”, “re-categorized” or “re-coordinated” based on experiences. By implication, the same symbol can be a different symbol if the contextual background of the other symbols that it occurs with is changed. The difference proposed by Clancey’s reformulation of the physical symbol system hypothesis suggests a focus in shift – from the symbol itself to the relations between conceptual categorizations that result in symbolic systems that are not static, but dynamic and changing. In a human being, knowledge is dynamic, because the “meaning” that a symbol captures changes as the conceptual categorizations that lie at its basis change. In a new experience, symbols are not simply retrieved from an older experience, copied and acted upon using descriptive rules. Perception, conception and action arise together as a physical re-activation
of categorizations. This he defines in terms of “structural couplings between concepts” and “re-activation and re-coordination” processes that can be simultaneous or sequential.

The important point is that, in a human being, a symbol is not an isolated atomic entity, but is an abstraction produced by a large body of perceptual and conceptual activations. These develop through experience. It is the dynamic relations activated in experiences that give rise to symbols that are important in capturing its meaning. Clancey’s theory provides insight into the fact that the “dynamic activations of associations, relations and patterns” view lies at the basis of the “rules between things” view. For an explicit “rule” to form, we first need a dynamic activated relation between the participating concepts based on all previous experiences that the reasoning agent has had. The “rule” is a higher order grounded result of lower order activations between concepts and percepts. At the lowest order, all combinatorial associations and relations are possible and plausible as perceptual and conceptual relations. Therefore, two experiences that have the same set of percepts and concepts could still be different, because the set of perceptual conceptual relations activated could be different.

As behavior criterion \textit{C4} for a design problem reformulation method, it should be able to \textit{model the dynamic relations existing between symbols in a design experience}. In formal terms, it should model \textit{the distributed map of association patterns between symbols in a design experience} – the mutual referencing between symbols that encode semantic meaning. That is, in addition to explicit logical / mathematical rules between symbols, the method should be able to “see” association patterns of occurrence between symbols. From a perceptual-conceptual basis, structural couplings exist between all symbols, whether or not they are explicitly related in the representation. Further, if the symbol set is changed, or explicit relationships between them are changed, then the “meaning” of one symbol and its importance changes relatively with respect to the others. The method should be able to “see” this.

3.3.2 \textbf{“Symbols aren’t simple”: Why a purely syntactical analysis reveals “meaning”}

A symbol is, in a way, the result of the “highest level” of abstraction produced by any species (Deacon, 1997) over perceptual conceptual dynamic activations in experiences. “Highest” does not imply a gradation of intelligence, only that the semiotic mode of reference and communication enables humans to demonstrate complex behaviors (like design or linguistic communication) that would not have been possible without the symbolic mode of reference. Clancey’s interpretations and reported experiments explore a sensory-motor perceptual level that occurs at levels before describing. Design is a high level cognitive activity that involves describing using symbols. To understand how symbols may reify knowledge in design representations, this section explores what kind of abstraction, extraction and encoding of
knowledge happens across experiences that involve symbols in large part (such as design experiences involving mathematical-symbolic modes of representation).

From a cognitive neuroscience perspective, Deacon (1997) uses C. S. Pierce’s theory of the three modes of reference – icons, indexes and symbols. An icon is a reference to an obviously perceived similarity between two things (a circle, coin and wheel), an index is an indication of some spatial or temporal correlation (a thermometer indicates temperature), while a symbol is an agreed upon conventional relationship between two things (a red traffic light denotes STOP). In this sense, an indexical association is one that exists between a symbol and the object in the world it is referring to (the one-to-one mapping discussed in Section 3.2; \( i \) to “actual” internal diameter of a cylinder). A symbolic association is one that exists between two symbols in the human mind (between \( i \) and \( s \), for instance). He proposes that reference is hierarchical. To be capable of indexical reference is to be already capable of iconic reference, and to be capable of symbolic reference is to be already capable of indexical reference. That is, to describe the hoop stress behavior in a hydraulic cylinder in symbolic terms (\( s = \pi p/2t \)), the mind must know beforehand the association between the physical hydraulic cylinder, pressure, or stress concepts and the corresponding mapping to their symbolic representations. Thus, symbolic reference is a higher order abstraction over sets and maps of indexical references, and the highest order abstraction over sets of all references, whether indexical or iconic.

This, however, is only the first proposition. His second, and stronger, proposition is presented by reporting extensive empirical results on actual symbol learning and grounding experiments performed with primates, the chimps Sherman, Austin and Lana. The proposition is this: “the learning problem associated with symbolic reference is a consequence of the fact that what determines the pairing between a symbol and some object or event is not a probability of their co-occurrence, but rather some complex function of the relationship that the symbol has to the other symbols…” (Deacon, 1997, p. 83). To situate this argument in design, say, the symbol is the design variable, and the event or object it is paired to is another variable. This pairing represents behavior expressed through a mathematical function. He is saying that a symbol does not lose its indexical association with the object it is referring to (“\( i \)” to the “actual” internal diameter of a cylinder – the symbol object one-to-one mapping) even though there is no direct physical referent present (an actual hydraulic cylinder does not exist yet – in design we are always referencing objects that are not built yet), because “the possibility of this link is maintained implicitly in the stable associations” between symbols (between design variables used in repeated design experiences). To continue with the example of the previous paragraph, the symbolic relationships between the stress, pressure, internal diameter and wall thickness implies a physics relationship (indexical) – but once established it can be used for learning, understanding and communicating the physics on a purely symbolic
plane quite apart from the physical. There is a kind of dual reference in operation – symbols refer to objects in the world (sense), and they also refer to each other (reference). The one-to-one symbol-object mapping is superseded by a word-word mapping, as in a dictionary – words derive their meaning from being described in terms of other words.

From an evolutionary perspective on how forms of symbolic communication and language have developed in human beings, it takes the indexical references (the symbol-object mapping between symbol and object in the world) to develop a symbolic system of representation, in a hierarchical way. However, after this symbolic system of representation has developed through extensive indexical interactions between symbol and object, the symbolic interactions become stronger than the indexical ones, even though the indexical ones may lie at the basis of developing the symbolic ones. When constructing a new reference for a new experience, the mutual reference between symbols is used to pick out the reference between objects and not vice versa. That is, as a designer, I will tend to think in terms of symbolic mathematical representations (Did I use this variable or constraint before? Will it be useful in its quadratic form, or should I use a linear approximation?) even though what I am actually doing is reasoning about physical objects and their behavior.

This directly implies that symbols exist contextually in a system – the power of one symbol to explain an indexical meaning about an object is distributed over its associations with all the other symbols that it exists with. Using language as the explanatory domain (I use the analogy between language and design from Chapter 2 as the bracketed words.), he says (Deacon, 1997, p. 83), “this referential relationship between words [design variables] – words [design variables] systematically referencing other words [design variables] – forms a system of higher order relationships that allows words [design variables] to be about indexical relationships [design structure and behavior], and not just indices in themselves. [This distribution of relations] is also why words [variables] need to be in context with other words [variables] in phrases and sentences [mathematical functions], in order to have any determinate reference [and potential “meaning”]. Their indexical power is distributed, so to speak, in the relationships between words [variables]. Symbolic reference derives from combinatorial possibilities and impossibilities, and we therefore depend on combinations both to discover it (during learning) [design] and to make use of it (during communication) [in our case, design representation].”

As behavior criterion C5 for the design problem reformulation method, it should be able to retrieve and infer not just the explicit one-to-one mapping between symbols (mathematically or logically defined relations in the problem representation) but also the implicit mapping between them (A and C are related to each other even though there is no explicit connection between them, not just because they individually co-occur with B, but also because of all the other co-occurrence patterns that A, B and C share with any other symbol in
the same context). Again, the same property of the “relative symbol” arises, as it did from the discussion on Clancey’s work – if the symbol set is changed, or the explicit relational mapping between the set of symbols in a design experience is changed, or the abstraction level to view it is changed, the same symbol becomes a different symbol, i.e. its encoded meaning changes and what it means in relationship to other symbols changes. The method should be able to “see” this, and more strongly “measure” this change.

3.3.3 Acquiring and inferring knowledge from episodes

A symbolic-mathematical design problem model is, in part, an abstract representation or encoding of a design experience. A design experience involves the use of information contained in other design experiences. A conscious exposition to a large number of examples and problems from the design domain and accumulation of experience are necessary factors for the development of expertise in design (Cross, 2004). As such, it can be claimed that design experiences, and by implication, design representations, involve, in part, the use of episodic memory and semantic memory. For example, studies on analogy making in design (Ball et al., 2004) have found that when designers apply abstract experiential knowledge in design problems, they use schema-driven analogizing (corresponding to semantic memory), and when they apply specific prior design problem knowledge to a new problem they use case-driven analogizing (corresponding to episodic memory).

Episodic memory is defined as the recall of some personal experience along with the spatial and temporal context in which the experience occurred, and semantic memory is defined as the use of general and invariant facts about the world as abstractions over episodic experiences (Mesulam, 1998). In empirical cognitive neuroscience, observation experiments using functional magnetic resonance imaging (fMRI) on human subjects report that the same part of the brain is shown as active when subjects are asked to remember details of some specific episodic past event, or construct a future episodic imaginary event (Schacter et al., 2007). This has led to claims about memory being a constructive process (Schacter & Addis, 2007), and the hypothesis that “the simulation of future episodes is thought to require a system that can flexibly recombine details from past events” (Schacter et al., 2007, p.659). That is, remembering past episodes and constructing future ones both require the retrieval of information, but more radically, the construction of a future task requires “that event details gleaned from various past events be flexibly recombined into a novel future event.” The study also hints that the same may be true for semantic memory (memory of general and invariant facts about the world) as it true for episodic memory.

As design always involves the construction of objects that do not exist yet, these findings suggest that previous design experience episodes will be useful for acting in future ones. A design representation is a dynamic trace of both episodic and semantic information. This
implies that when previous design episodes are used for future episodes, the ways in which previous episodes are used involve retrieving multiple combinatorial patterns from previous episodes. Often, there is no one dominant formulation, and several multiple formulations and reformulations may be formed in the mind. Expertise development, in part, could be a measure of how well a designing agent is able to tear apart and re-construct immediately preceding episodic information in order to act in the current design experience.

Reformulation is a special example of a design task wherein the immediately preceding (previous) episodic information is used to infer future episodic construction within the same design experience. Symbols in a design representation are “events”. These occur in functions or design interaction / dependency mappings with each other. Therefore, a specific mapping between symbols, i.e. an analytical functional or non-analytical mapping between symbols, defines an episode. Note that a symbol occurs in many specific episodes. Therefore, a symbol is, in a way, a semantically invariant abstraction. The way it comes together with other symbols is episodic – i.e. a function or dependency mapping is usually unique in a design experience. Thus, a design representation contains episodic-semantic information. Measuring expertise in design modeling and reformulation will be a measure of how well a designing agent is able to tear apart and re-construct immediately preceding episodic information in order to predict immediately following episodic information in an interactive way.

As behavior criterion C6 for the design problem reformulation method, it **should be able to use the episodic information in a design representation as the basis for reformulating a design and constructing alternate design representations.**

C4, C5 and C6 establish the cognitive-neuroscience based design theory criteria for the method.

### 3.4 Summary: Additional performance criteria for method

In summary of the above analysis and the analysis in Chapter 2, the following are behavior criteria identified for the design problem reformulation method REIFORM.

**Design methodology criteria:**

*C1:* The method should be knowledge-lean, i.e. it should not require high levels of knowledge engineering.

*C2:* The method should be training-lean, and should use the knowledge acquired in a single design experience to act upon the same design experience and reformulate the problem.

*C3:* The method should allow heuristic problem exploration by inferring multiple interpretations from a single representation, and should be general, i.e. applicable across problems of varying complexity from various domains using different representational forms.

**Cognitive-neuroscience based design theory criteria:**
C4: The method should be able to model the distributed map of association patterns between symbols in a design experience in a dynamic manner.

C5: The method should be able to retrieve and infer not just the explicit one-to-one mapping between symbols, but also the implicit mappings.

C6: The method should be able to use the episodic relationships in a design representation as the basis for reformulating a design and constructing future design representations.

The next chapter describes the REIFORM method in detail, and demonstrates it on an illustrative analytically stated design problem. Chapters 5, 6 and 7 illustrate and develop the method for various types of problem reformulation tasks. Chapter 9 presents an assessment of method performance against these 6 criteria.
Chapter 4
Method: REIFORM

*The SVD in linear algebra...is one more case, if further convincing in necessary, in which mathematics gets the properties right – and the applications follow.*

Gilbert Strang, *The Fundamental Theorem of Linear Algebra*

This chapter presents the REIFORM method developed for automating design problem reformulation tasks. The method works by inductively inferring the major associative patterns from a local distribution of relationships between events and episodes in a design experience. Figure 4.1 shows a design problem formulation example that will be used to demonstrate the method in this chapter. The terminology to be used is exemplified in the figure. As a quick review of this terminology – an *event* is a design symbol that signifies a structure or behavior variable or parameter in an analytical formulation, or a structure or behavior variable or design component in a non-analytical formulation. \( x_1, x_2, \ldots, x_6 \) are examples of events in the problem formulation of Figure 4.1. An *episode* is a relationship that connects two events. \( h_1, h_2, \ldots h_8 \) are all examples of episodes in Figure 4.1.

Event \( i \) and event \( j \) (variables, parameters, design components) are *similar* if they occur together in the same episodes or if they occur, together or individually, with any other common events. Episode \( i \) and episode \( j \) are *similar* if they contain the same or overlapping sets of common events. Intuitively, the “strength” of similarity depends upon the “degree” of overlap. For example, in Figure 4.1, while \( h_2 \) and \( h_3 \) are very similar, \( h_2 \) and \( h_5 \) are dissimilar. Similarity, in this work, is not a Boolean concept, where two things are either similar or dissimilar. It is a concept that is best described by thinking of a continuously varying range of values, where the value describes the “strength” of similarity between two things.
One set of events and/or episodes expressed as a symbolic-mathematical design model defines a design experience for REIFORM. The hypothesis is that the seat of semantic meaning of the design lies, in part, in the explicit or implicit syntactic relationship patterns between events and episodes. Therefore, REIFORM can acquire and infer semantic meaning of a design from its syntactic representation as represented by the major associative patterns between events and episodes. REIFORM is scoped for a class of reformulation tasks that may be performed by processing sets of associative transformations between events and episodes in design experiences, and can be performed with purely syntactical analyses on formulation examples.

The analytically formulated problem of Figure 4.1 is used as an example to demonstrate each step of REIFORM. This chapter describes the skeletal steps of REIFORM without describing the specific ways in which it can be used for problem reformulation. The method structure remains the same for all subsequent problem reformulation tasks, though the ways in which it is used and interpreted varies. Chapters 5, 6 and 7 demonstrate the application of the method for specific reformulation tasks.

4.1 Step I: Data Representation

The first step is to convert an analytical or non-analytical design problem model into an occurrence matrix.

4.1.1 Analytical formulations

An analytical formulation is a problem formulation in which variables and parameters are represented as symbols, and mathematical functions describe relationships between them. Design optimization problems are commonly stated as analytical formulations in the following general form (Papalambros & Wilde, 2000):

Min \( f(x, p) \)
Sub to
\[ g(x, p) \leq 0 \]
\[ h(x, p) = 0 \]
\[ x, p \in \chi \subseteq \mathbb{R}^n \] (4.1)

Here, \( x \) is the vector of design variables and \( p \) is the vector of design parameters that are kept fixed for one design model. These belong to a subset \( \chi \) of the real space \( \mathbb{R}^n \). The vector \( f \) is the vector of objective functions; for a single objective problem this will be \( f \). The vectors \( g \) and \( h \) are inequality and equality constraints. The objective in the optimal design problem is to minimize some behavior and find the optimal set of values for variables that produce this behavior. For a general design problem (Michelena & Papalambros, 1997), the objective is to find a feasible (satisficing) solution, and a feasible set of values for variables that produce this behavior – the solution may or may not be an optimal one. The occurrence matrix can be generated both for the optimal design problem or the general design problem statements.

The occurrence matrix generated from an analytical formulation captures the local co-occurrence relationships between variables and parameters and their functional relationships, i.e. between events and episodes. As the first step of REIFORM, the mathematical problem model is converted into an occurrence matrix \( A \) as follows: For a design problem model that has \( m \) variables and parameters and \( n \) functions, the occurrence matrix \( A \) has \( m \) rows, each representing a variable/parameter, and \( n \) columns, each representing a function (objective/constraint). Each matrix entry \( A_{ij} \) is either a 1 or a 0 depending upon whether or not a particular variable or parameter \( i \) occurs in objective or constraint \( j \). Figure 4.1 shows an example formulation for a problem (Michelena & Papalambros, 1997) and Figure 4.2 the respective occurrence matrix \( A \).

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( f )</th>
<th>( h_1 )</th>
<th>( h_2 )</th>
<th>( h_3 )</th>
<th>( h_4 )</th>
<th>( h_5 )</th>
<th>( h_6 )</th>
<th>( h_7 )</th>
<th>( h_8 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_2 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( x_3 )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( x_4 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( x_5 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( x_6 )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( x_7 )</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( x_8 )</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 4.2: The occurrence matrix for Figure 4.1

4.1.2 Non-analytical formulations

A non-analytical formulation is a design formulation in which design elements are represented as symbols and relationships between them are captured as Boolean mappings in a matrix: there exists or does not exist a functional, interaction or dependency relationship between two design variables, parameters or system components. In some cases, the mapping defined can also be other than Boolean. In general, non-analytical models can either be
described as a Functional Dependence Table (FDT) (Wagner & Papalambros, 1993a), or a Design Structure Matrix (DSM) form (Pimmler & Eppinger, 1994). Other forms of non-analytical representations, like the Multiple Domain Matrix (Lindemann et al., 2009) exist. Though this thesis does not demonstrate method application on examples from all the possible non-analytical representations, it is easy to see that the method is applicable on any representation that captures design interactions in matrix form. In the case of multiple-domain matrices, for example, the analysis would have to be applied onto each matrix to observe the resulting decomposition.

In an FDT formulation, relationships are represented in a Boolean binary (1-0) mapping between variables and functions. The mapping, similar to Figure 4.2, shows whether a relationship exists between a function and a variable. Usually, the FDT is rectangular in form. If the numbers of design variables and functions are equal, this would lead to a square matrix. The FDT may be derived from analytical formulations or numerical simulation results. For example, Michelena and Papalambros convert the analytical problem in Figure 4.1 into the FDT form.

The FDT form is the transpose of the form used in this thesis: in an FDT, the rows represent design relationships (functions, attributes, etc.) and the columns represent variables. Therefore, to generate an occurrence matrix $A$ from an incidence matrix, the matrix is simply transposed.

In a DSM, the matrix relationships are represented in a Boolean mapping between the same elements in rows as well as columns. These elements could be design variables or system components. The relationships are design dependencies of some kind, for example, spatial adjacency or material/information/energy exchanges. The matrix entries could be any number (not necessarily binary 1 or 0) and could be representative of the positive or negative strength of relationships.

The DSM representation itself is the occurrence matrix. However, in a DSM, the diagonal entries, i.e. the relationship of an element with itself, is not defined or left blank. In the occurrence matrix $A$, the diagonal entries are the maximum positive value being used in the representation. This is based on the assumption that a design element has the maximum interaction with itself (for example, it is closest to itself in terms of spatial adjacency or has the highest material interaction with itself). This definition was necessary to ensure that the method produces consistent results.

Note that one major difference between the occurrence matrix generated from analytical/FDT form and the occurrence matrix generated from a DSM form is that, in the former, the rows and elements represent a mapping between two different types of elements (variables and functions, as event-episode mappings), while in the latter, the rows and elements represent a mapping between the same types of things (event-event mappings).
Step II: Performing SVD on the occurrence matrix

The second step is to calculate the singular value decomposition (SVD) of the occurrence matrix $A$.

SVD takes a general rectangular matrix $A$ with $m$ rows and $n$ columns and decomposes it into a product of three matrices, $A = USV^T$, where $U(m \times m)$ and $V(n \times n)$ are the left and right orthogonal matrices and $S(m \times n)$ is a rectangular matrix with non-negative singular values on the diagonal in order of decreasing magnitude. The number of singular values is $r$, where $r$ is the rank of $A$. The mathematical idea (Figure 2.7; (Strang, 1993, 2003)) is as follows: the row space of $A$ is $r$-dimensional and inside $\mathbb{R}^n$, and the column space of $A$ is $r$-dimensional and inside $\mathbb{R}^m$. We choose special orthonormal bases $V = (v_1, v_2, \ldots, v_r)$ for the row space, and $U = (u_1, u_2, \ldots, u_r)$ for the column space, such that $Av_i$ is in the direction of $u_i$. $s_i$ provides the scaling factor and $Av_i = s_iu_i$. In matrix form, this becomes $AV = US$ or $A = USV^T$. $A$ is thus the linear transformation that carries orthonormal basis $v_i$ from space $\mathbb{R}^n$ to orthonormal basis $u_i$ in space $\mathbb{R}^m$. Neglecting the null spaces, $A(m \times n) = U(m \times r) * S(r \times r) * V(r \times n)$. Another way to say the same thing is that a general rectangular matrix $A$ has been diagonalized into two independent spaces described by orthogonal bases $U$ and $V$, and related to each other by the magnitudes of the singular values. Figure 4.3 shows the results of SVD performed on the occurrence matrix in Figure 4.2.

The matrix $A$ represents the explicit design relationships that the designer chooses to capture in the problem model, i.e. how each symbol occurs in a function and the inter-relations between symbols across all functions. Consider what happens mathematically in this SVD step using the FDT form as an example (all the steps remain equivalently valid for the DSM form). In the matrix $A$, the rows represent variables and parameters and the columns represent functions. Each row in matrix $A$ is a measure of how each variable or parameter occurs across the functions (objectives and constraints). Each column in matrix $A$ is a measure of which variables or parameters are contained in each function. Each of the $m$ variables or parameters is/is not related to $n$ functions, i.e. the row space of $A$ is $r$-dimensional inside $\mathbb{R}^n$, the null space of $A$ is $(n-r)$-dimensional. Each of the $n$ functions contains / does not contain any of the $m$ variables and parameters, i.e. the column space of $A$ is $r$-dimensional inside $\mathbb{R}^m$, the null space of $A^T$ is $(m-r)$-dimensional.

4.2.1 Defining the event (variable, parameter, design component) space

When SVD is performed, $U$ contains the same number of rows as the original matrix $A$, i.e. corresponding to the $m$ number of variables or parameters, but a new set of $r$ columns that are independent of each other. These are the new derived orthonormal left singular vectors; each of them is independent of each other, cannot be described in terms of any of the other vectors.
Therefore, the $i^{th}$ variable/parameter is now described by the $i^{th}$ row of $U$ with $r$ components in an $r$-dimensional space. For example, observe from Figure 4.3 that the rank $r$ of the matrix $A$ is 8, since there are 8 singular values. The variable $x_1$ (the first row of $U$) is described by the first components of the 8 left singular vectors. This can be thought of as $x_1$ is 0.53 parts of abstract vector 1, -0.41 parts of abstract vector 2, and so on. $U$ is the abstract “event” space or “variable-parameter” space.

$$A = USV^T$$

$U$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5325</td>
<td>-0.4123</td>
<td>-0.0470</td>
<td>0.4538</td>
<td>0.2293</td>
<td>-0.0981</td>
<td>0.4905</td>
<td>-0.1865</td>
</tr>
<tr>
<td>2</td>
<td>0.6072</td>
<td>0.3675</td>
<td>-0.1350</td>
<td>0.0226</td>
<td>-0.0011</td>
<td>-0.0335</td>
<td>-0.5539</td>
<td>-0.4116</td>
</tr>
<tr>
<td>3</td>
<td>0.1822</td>
<td>-0.0853</td>
<td>-0.5601</td>
<td>-0.2629</td>
<td>0.5238</td>
<td>-0.0421</td>
<td>-0.1285</td>
<td>0.5330</td>
</tr>
<tr>
<td>4</td>
<td>0.2270</td>
<td>-0.6182</td>
<td>0.1260</td>
<td>-0.0751</td>
<td>-0.4912</td>
<td>-0.2532</td>
<td>-0.3946</td>
<td>0.2886</td>
</tr>
<tr>
<td>5</td>
<td>0.4534</td>
<td>0.3522</td>
<td>0.1079</td>
<td>-0.0578</td>
<td>-0.3997</td>
<td>0.3228</td>
<td>0.3429</td>
<td>0.4837</td>
</tr>
<tr>
<td>6</td>
<td>0.0620</td>
<td>-0.2236</td>
<td>-0.4093</td>
<td>-0.6246</td>
<td>-0.2876</td>
<td>0.1952</td>
<td>0.2928</td>
<td>-0.4266</td>
</tr>
<tr>
<td>7</td>
<td>0.1428</td>
<td>-0.2769</td>
<td>0.5105</td>
<td>-0.2485</td>
<td>0.3760</td>
<td>0.6372</td>
<td>-0.1746</td>
<td>-0.0408</td>
</tr>
<tr>
<td>8</td>
<td>0.1825</td>
<td>0.1530</td>
<td>0.4579</td>
<td>-0.5131</td>
<td>0.2192</td>
<td>-0.6125</td>
<td>0.2159</td>
<td>-0.0215</td>
</tr>
</tbody>
</table>

$S$

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.9317</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2.2866</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1.7481</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.5941</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.3735</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.69813</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.46254</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.27204</td>
</tr>
</tbody>
</table>

$V$

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.1110</td>
<td>-0.5555</td>
<td>0.5540</td>
<td>-0.4778</td>
<td>0.4333</td>
<td>0.0353</td>
<td>0.0893</td>
<td>-0.2290</td>
</tr>
<tr>
<td>2</td>
<td>0.3078</td>
<td>-0.5772</td>
<td>0.3384</td>
<td>0.0817</td>
<td>0.0830</td>
<td>0.4096</td>
<td>-0.1701</td>
<td>0.2175</td>
</tr>
<tr>
<td>3</td>
<td>0.4240</td>
<td>0.4026</td>
<td>0.2465</td>
<td>-0.3402</td>
<td>-0.1322</td>
<td>-0.4629</td>
<td>0.0106</td>
<td>0.2231</td>
</tr>
<tr>
<td>4</td>
<td>0.5434</td>
<td>0.1534</td>
<td>-0.0424</td>
<td>0.2664</td>
<td>-0.1249</td>
<td>0.2739</td>
<td>0.6043</td>
<td>-0.3908</td>
</tr>
<tr>
<td>5</td>
<td>0.4569</td>
<td>-0.0444</td>
<td>-0.4245</td>
<td>0.1377</td>
<td>0.5474</td>
<td>-0.2467</td>
<td>-0.4149</td>
<td>-0.2463</td>
</tr>
<tr>
<td>6</td>
<td>0.9866</td>
<td>-0.3711</td>
<td>-0.1609</td>
<td>-0.4390</td>
<td>-0.5670</td>
<td>-0.0831</td>
<td>-0.2199</td>
<td>-0.5073</td>
</tr>
<tr>
<td>7</td>
<td>0.2591</td>
<td>-0.4543</td>
<td>0.0463</td>
<td>0.2376</td>
<td>-0.1907</td>
<td>-0.5032</td>
<td>0.2075</td>
<td>0.3675</td>
</tr>
<tr>
<td>8</td>
<td>0.3618</td>
<td>0.3351</td>
<td>-0.0155</td>
<td>-0.0183</td>
<td>-0.2918</td>
<td>0.4144</td>
<td>-0.4562</td>
<td>0.3020</td>
</tr>
<tr>
<td>9</td>
<td>0.0833</td>
<td>-0.1362</td>
<td>-0.5546</td>
<td>-0.5567</td>
<td>0.1720</td>
<td>0.2193</td>
<td>0.3854</td>
<td>0.3912</td>
</tr>
</tbody>
</table>

**Figure 4.3: SVD of matrix A**

### 4.2.2 Defining the episode (function/interaction) space

Similarly, when SVD is performed, $V^T$ contains the same number of columns as the original $A$, i.e. corresponding to the $n$ number of columns, but a new set of $r$ rows that are independent of each other. These are the new derived orthonormal right singular vectors; each of them is independent of each other, cannot be described in terms of any of the other vectors. Therefore, the $j^{th}$ function is now described by the $j^{th}$ column of $V$ with $r$ components in an $r$-dimensional space. For example, observe from Figure 4.3 that the rank $r$ of the matrix $A$ is 8, i.e. there are 8 singular values. The function $f$ (the first row of $V$) is described by the first components of the 8 right singular vectors. This can be thought of as $f$ is 0.11 parts of abstract vector 1, 0.30 parts of abstract vector 2, and so on. The 9th row is neglected; it goes to the null space. $V^T$ is the abstract “episode” space or “function/interaction” space.
4.2.3 Defining the US and $SV^T$ spaces

Consider the following argument for these two spaces:

1. The form $AV = US$ implies that a linear transformation $A$ of the independent “function/interaction” space is equal to a linear combination of the independent “variable-parameter” space, where the singular values provide the scaling linear combination factors.

2. The form $A^T U = SV$ implies that a linear transformation $A$ of the independent “variable-parameter” space is equal to the linear combination of the independent “function/interaction” space, where the singular values provide the scaling linear combination factors.

Thus, $US$ and $SV^T$ spaces describe a scaled event and a scaled episodic space respectively in $r$-dimensions.

4.2.4 Geometric interpretation of mathematics

Consider now the conceptual or geometric interpretation of the mathematics. SVD takes an original matrix of local explicit relationships and produces a new set of abstract vectors that are independent of each other and related by the singular values. The linear combinations (in terms of singular values) of these vectors now describe the variables, parameters and functions uniquely as vectors in an $r$-dimensional space – a description that takes into account how each of them is related to all the others depending on the association patterns in the occurrence matrix, i.e. each variable-parameter or function vector can now be “plotted” in an $r$-dimensional space as a single point. $US$ and $SV^T$ are special linear combinations because the singular values capture the most important associative patterns in the data in a decreasing order of magnitude, i.e. the largest ones capture the most important relationships and so on.

The occurrence matrix $A$ contains measurements of the local explicit contextual occurrence of variables and parameters in functions. SVD converts this into two mathematical spaces where this distribution is described in terms of independent components that measure all possible linear combinations/inter-correlations between the elements in the original co-occurrence matrix, but are themselves uncorrelated to each other. Whereas the original matrix $A$ contained only direct associations between variables, parameters and functions, SVD computes linear combinations of data using every cell entry in the matrix to produce these new vectors to represent a variable, parameter and function. SVD takes the local explicit occurrence relationships and produces something like a linear “global” association map, and re-represents this with two spaces where each independent dimensional vector is now a measure of how each variable, parameter or function is related to all the others. An important point to note is that if only one entry in the occurrence matrix is changed, then this is enough to produce changes in all of the components.
4.3 Step III: Dimensionality reduction

The third step is to perform a dimensionality reduction on the decomposed matrices, U, S and V to produce a truncated linear least squares approximation of A.

From the U, S and V matrices in Figure 4.2, if we retain the first k singular values in S and compute a truncated approximation of A as \( A'(m \times n) = U(m \times k) \times S(k \times k) \times V^T(k \times n) \), it will be a least squares approximation of A. This is well established in linear algebra that SVD analysis and retention of the first k singular vectors produces an optimal k-rank least squares approximation of the original matrix (refer to (Kalman, 1996) or (Strang, 2003) for derivations and proofs). For example, preserving the first 2 dimensions for the example SVD in Figure 4.3 produces the k-reduced truncated matrix shown in Figure 4.4.

\[
A' = U_{(m \times k)} \times S_{(k \times k)} \times V^T_{(k \times n)}
\]

### Figure 4.4: k-reduced approximation for matrix A, k = 2

4.3.1 Interpreting approximations to US and SV^T spaces:

An inductive argument then, developing from the argument developed in Section 4.2.3 in (1) and (2), is that an approximation of the linear transformation A of the independent variable (or function/interaction) space is equal to an approximation of the linear combination of the independent function/interaction (or variable-parameter) space, where the first k singular values provide the linear combination scaling factors. Approximations of the US and SV^T spaces implies choosing a smaller number of singular values and number of components from the orthonormal vectors to produce the linear combination. Logically, any linear combination that is an approximation of the original US and SV^T spaces is a valid approximation of the original design problem representation, because the orthonormal vectors and singular values are produced by using the same space.
A dimensionality reduction implies that instead of using \( r \) dimensions or abstract vectors to describe a variable, parameter or function, a lower number \( k \) is used. What this means in abstract terms is that each variable, parameter or function is re-represented as a linear combination of only \( k \) dimensions instead of \( r \) and is a best guess (least square approximation) on the relationships shared between variables, parameters or functions.

### 4.3.2 How do the approximations capture implicit information?

The special structure of the SVD decomposition says that the decomposition can be viewed in terms of \( r \) rank one matrices. That is, the best rank 1 approximation to \( A \) is the matrix \( u_1 s_1 v_1^T \), using the first singular value and the first left and right singular vectors (Strang, 2003).

Similarly, the best rank 2 approximation is \( u_1 s_1 v_1^T + u_2 s_2 v_2^T \) and at \( r \) is \( \sum_{i=1}^{r} u_i s_i v_i^T \). The reason that the approximations are interesting is because of the way in which they capture associations through induction. The singular values are arranged in an order of decreasing magnitude, i.e. the largest first. Therefore, the rank \( k \) approximations capture the major or strongest association patterns from the matrix. The effect that this produces is that when a lower number of singular values is being considered to approximate (i.e., “guess”) whether variable \( i \) appeared in constraint \( j \), the approximation will capture the major association patterns and ignore the smaller ones. The removal of singular values that are not considered in the approximation have the effect of removing any “noise” in the original formulation such that the approximated representation contains only the dominant relations. The relationships between variables and constraints that mutually share a large number of relationships must be scaled up so that the linear combination is as close as possible to the original. The relationships between variables and constraints that do not share mutual associations will be scaled down. Because this is an optimized least squares approximation, the best approximation will have to scale the original relationships up or down to bring it closest to the original distribution. In doing so, for example, it will have to show a higher than 0 relationship between a variable \( i \) and a constraint \( j \) even if the explicit occurrence matrix relationship is 0, if variable \( i \) occurs with other variables and in other constraints that have a high mutual occurrence relationship with constraint \( j \). Therefore, this dimensionality reduction step is the seat for finding implicit relationships that are not obvious or evident from the original occurrence matrix or the problem representation.

Recall that the definition of semantic similarity developed in Chapter 2 and in the beginning of this chapter was that sets of events that tend to appear together in episodes within an experience have more semantic (design) connections with each other than sets of events that do not tend to co-occur in the same episodes within an experience. The first \( k \) singular values capture the most important association patterns in the data, in decreasing order.
of magnitude. Approximations to these spaces imply using different linear combinations, that is, different numbers of singular values. It is known that in decreasing order of magnitude, the singular values capture the most important associative patterns. Therefore, the approximations, that are linear least square approximations, will overplay the most important associative patterns and underplay the rarer ones. That is, variables and functions that mutually share high relations will be strengthened in the earlier approximations, and those that do not will be weakened.

This means that if two events or episodes occur strongly with each other or with other common events and episodes (according to the similarity definition, are “similar” to each other) then they would tend to be placed close to each other in this re-represented approximation space. Since the components of the US and SVT spaces are vectors in an original r or reduced k dimension space, this interaction strength relationship is, in effect, converted into a distance relationship. Any points lying close in this re-represented k-dimensional space are “semantically similar”, and any points lying far are “dissimilar”. This is useful for problem reformulation tasks, because intuitively, this step will capture how an event or an episode affects, by relative association strength, all the other events and episodes. As later chapters will demonstrate, reformulation tasks such as problem decomposition or topology planning can be performed with this definition. At some k values, this step intuitively minimizes the distance between strongly connected variables and functions and pulls them closer in this re-representation space due to the k largest singular values that capture the strongest relations. At the same time, it maximizes the distance between the not strongly connected variables and functions and pushes them far in space, again due to disregarding the (r-k) smallest singular values treating the weaker association patterns as “noise”.

4.3.3 Example demonstration

For example, consider from Figure 4.1 that the variables x₁ and x₄ appear in function h₁, and x₂ and x₅ appear in functions h₂, but x₁, x₄, x₂ and x₅ do not appear directly in objective function f. It is intuitive and obvious from our knowledge of algebra that f could just be re-written in terms of x₁, x₂, x₄ and x₅, but such an “intuition” is not something that can be taught to an automated system without an explicit rule about algebraic substitution. Thus, in the original matrix A (Figure 4.2), the data entry A₁₁ (relationship between x₁ and f) is 0, and the data entry A₁₂ (relationship between x₁ and h₁) is 1. These are explicit syntactic relationships. However, it is quite obvious that semantically x₁ and f are correlated in a latent way, because h₁ appears in f. Observing the same two entries in the 2D-truncated A’ (Figure 4.4) shows that now A’₁₁ is 0.22513 and A’₁₂ is 1.0204, showing that in this case, the 0 relationship has scaled up to 0.22 and the 1 relationship has become stronger. Absolute 0-1 relationships have scaled
up between the three quantities. That is, the dimensionality reduction step says that using only 2 dimensions to make an approximation or a best guess reveals that variable $x_1$ appears in $f_{0.22}$ times, and variable $x_1$ appears in $h_1_{1.0204}$ times. Similarly, note from Figure 4.1 that data entry $A_{43}$ (the relationship between $x_4$ and $h_2$) is 0. Due to other relationships (and the reason for this will be confirmed in the next chapter where this problem is used to demonstrate design decomposition) in the 2D approximation, note from Figure 4.4 that data entry $A′_{43}$ is -0.2825. The approximation says that the best guess is that $x_4$ appears in $h_2 - 0.2825$ times – an original absolute 0 relationship is changed to an even lower value. In general, note from Figure 4.4 how all the 1s and 0s have been scaled higher or lower in terms of scaled up or scaled down mutual relations between the entries. This implies that original entries where a 0 or a 1 signified no relationship or a perfect relationship between the two respective quantities are now changed to show a higher or lower (positive or negative) relationship, depending on occurrence relationships shared by one element with all the others.

### 4.3.4 How implicit relationships change over $k$ values

As the $k$ values are further increased and varied, the approximations $A′$ will be increasingly accurate approximations to the original occurrence matrix $A$. At $k=r$, the matrix will become the same again, $A′ = A$. Thus, note these two points:

1. The values in the $A′$ approximations approach the limit set by the values in $A$ as the $k$ values are increased. The implicit information is highest in the lowest approximations.
2. If distances were to be computed between the vectors or points representing the events and episodes in $k$-dimensional space ($A′$), then these distances would also approach the limit set by the distances between events and episodes in $r$-dimensional space ($A$).

Ideas for reformulating the problem may lie in some of these implicit relationships that the approximation shows. Table 4-1 shows for the example problem how the $k$-reduced approximations approach the limit set by the original matrix $A$ as the $k$ values are increased. The first row “OM” shows the explicit Occurrence Matrix entries that $x_1$ shares with the objective function $f$ and the functions $h_1$ to $h_8$. The other rows show the approximations computed at different $k$ values. Note that as the $k$ values are increased, the approximations increasingly approach the limit set by the $k = 8$, i.e. full rank space. In the full-rank space, only the explicit information, as specified in the occurrence matrix is returned.

Sections 4.4 and 4.5 describe how unsupervised similarity measurements can be performed by computing cosine measurements between events and episodes in these $k$-reduced spaces. A lower cosine measurement implies a low implied interaction strength between events and episodes in the $k$ space, a high cosine measurement implies a high interaction. Table 4-2 shows that similar to the approximations themselves, the cosine distances also approach a limit set by the full rank space. Again, setting $x_1$ as the query
variable, cosines are computed between \( x_1 \) and all other functions. The first row shows the original occurrence matrix entry. The other rows show cosine distances that \( x_1 \) shares with all other functions in the respective \( k \)-reduced spaces. The last row at full rank \( k = 8 \) corresponds to the original occurrence matrix entries. As \( k \) values are increased, the distances approach the limit set by the full rank space.

<table>
<thead>
<tr>
<th>Query: ( x_1 )</th>
<th>( f )</th>
<th>( h_1 )</th>
<th>( h_2 )</th>
<th>( h_3 )</th>
<th>( h_4 )</th>
<th>( h_5 )</th>
<th>( h_6 )</th>
<th>( h_7 )</th>
<th>( h_8 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( k=2 )</td>
<td>0.23</td>
<td>1.02</td>
<td>0.29</td>
<td>0.70</td>
<td>0.76</td>
<td>0.50</td>
<td>0.83</td>
<td>0.25</td>
<td>0.26</td>
</tr>
<tr>
<td>( k=3 )</td>
<td>0.18</td>
<td>0.99</td>
<td>0.27</td>
<td>0.71</td>
<td>0.79</td>
<td>0.51</td>
<td>0.83</td>
<td>0.25</td>
<td>0.30</td>
</tr>
<tr>
<td>( k=4 )</td>
<td>-0.17</td>
<td>1.05</td>
<td>0.02</td>
<td>0.90</td>
<td>0.89</td>
<td>0.20</td>
<td>1.00</td>
<td>0.24</td>
<td>-0.10</td>
</tr>
<tr>
<td>( k=5 )</td>
<td>-0.03</td>
<td>1.08</td>
<td>-0.02</td>
<td>0.86</td>
<td>1.06</td>
<td>0.02</td>
<td>0.94</td>
<td>0.15</td>
<td>-0.05</td>
</tr>
<tr>
<td>( k=6 )</td>
<td>-0.03</td>
<td>1.05</td>
<td>0.01</td>
<td>0.84</td>
<td>1.08</td>
<td>0.02</td>
<td>0.97</td>
<td>0.12</td>
<td>-0.06</td>
</tr>
<tr>
<td>( k=7 )</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>( k=8 )</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Table 4-1: The entries in \( A' \) approach the limit set by the entries in \( A \) as the \( k \) values are increased; here, the rows show the matrix entries from the approximations for variable \( x_1 \) with all the other functions.

<table>
<thead>
<tr>
<th>Query: ( x_1 )</th>
<th>( f )</th>
<th>( h_1 )</th>
<th>( h_2 )</th>
<th>( h_3 )</th>
<th>( h_4 )</th>
<th>( h_5 )</th>
<th>( h_6 )</th>
<th>( h_7 )</th>
<th>( h_8 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( k=2 )</td>
<td>0.99</td>
<td>0.91</td>
<td>0.39</td>
<td>0.73</td>
<td>0.90</td>
<td>0.76</td>
<td>0.92</td>
<td>0.40</td>
<td>0.94</td>
</tr>
<tr>
<td>( k=3 )</td>
<td>0.29</td>
<td>0.84</td>
<td>0.36</td>
<td>0.73</td>
<td>0.81</td>
<td>0.74</td>
<td>0.92</td>
<td>0.40</td>
<td>0.39</td>
</tr>
<tr>
<td>( k=4 )</td>
<td>0.00</td>
<td>0.80</td>
<td>0.20</td>
<td>0.75</td>
<td>0.80</td>
<td>0.33</td>
<td>0.92</td>
<td>0.36</td>
<td>0.04</td>
</tr>
<tr>
<td>( k=5 )</td>
<td>0.07</td>
<td>0.80</td>
<td>0.18</td>
<td>0.72</td>
<td>0.78</td>
<td>0.18</td>
<td>0.86</td>
<td>0.29</td>
<td>0.07</td>
</tr>
<tr>
<td>( k=6 )</td>
<td>0.06</td>
<td>0.78</td>
<td>0.18</td>
<td>0.71</td>
<td>0.78</td>
<td>0.18</td>
<td>0.84</td>
<td>0.28</td>
<td>0.06</td>
</tr>
<tr>
<td>( k=7 )</td>
<td>0.07</td>
<td>0.77</td>
<td>0.18</td>
<td>0.71</td>
<td>0.75</td>
<td>0.17</td>
<td>0.84</td>
<td>0.26</td>
<td>0.07</td>
</tr>
<tr>
<td>( k=8 )</td>
<td>0.07</td>
<td>0.77</td>
<td>0.18</td>
<td>0.71</td>
<td>0.75</td>
<td>0.18</td>
<td>0.84</td>
<td>0.26</td>
<td>0.07</td>
</tr>
</tbody>
</table>

Table 4-2: Distances between \( x_1 \) and all other functions in \( k \)-reduced spaces approach the distances between \( x_1 \) and all other functions in the \( k=r \) space

Observe that implicit relationships are returned using the largest singular values. As the \( k \) values are increased, the approximations begin to return only the explicit relationships. This observation is relevant to the finding to well-formed reformulation solutions in the later chapters, and the choice of \( k \) is therefore an important research question. The example presented here is used to demonstrate design decomposition in the next chapter. The \( k = 2 \) approximation led to the correct reformulation solution. For more complex problems (larger size / more complex interaction structure), one may experiment with a higher number of dimensions to retain. Chapter 8 presents parametric studies and heuristics on how to choose \( k \) values that return well-formed reformulation solutions.

Using the same observation, it is also easy to see that no reformulation is possible using only the original occurrence matrix entries. For example, at \( k = 8 \), no implicit information is returned. In a design representation, many sorts of relationships are possible, but the designer chooses only some of them. A reformulation of the problem implies changing some of these explicit relationships into another form. How is this to be done? An explicit representation has
inherent multiple implicit patterns deriving from these explicit relationships. These are latent implied patterns and are not observed from the original matrix, but these may be the seat of a reformulation. The intuition is that the implicit relationships provide something akin to “design freedom” – more numbers of relationships are plausible than what is shown in the explicit formulation. Different $k$ values will bring out different and gradually varying “intensities” of relationships between the events and the episodes, and suggest different implicit relationships. A range of these will be useful for reformulating the problem because they identify new formulations which are approximately the same as the original formulation, but which may have useful properties such as better modularity or they are well-formulated for numeric or symbolic optimization. While the illustrative example discussed here may be obvious, later chapters will show how this property of observing latent relationships between variables and functions becomes important in large scale problems or in problems where such relationships cannot be observed directly because of the complexity of interactions.

4.4 Vector representation of dimensionality reduction: geometric interpretation

It is helpful at this point to visualize how the $k$ reduction scales the variable-parameter space and the function space. From the original decomposition $A = USV^T$ and the general theory of SVD, consider the two equations $AV = US$ and $A^TU = SV$. As discussed earlier, the $US$ space is the scaled event or occurrence (variable-parameter) space and the $SV^T$ space is the scaled episode (function/interaction) space. Formally, let event (or occurrence) space $O = US$, and episode space $E = SV^T$. Then event or occurrence space $O (m \times k)$ has $m$ rows, each having $k$ components. Thus, we can rewrite:

$$O = (o_1, o_2, \ldots, o_m) \quad (4.2)$$

where each $o_i = \{o_{i1}, o_{i2}, \ldots, o_{ik}\}$ implies event $i$ represented as a $k$ component vector in $k$-dimensional space. Similarly, $E (k \times n)$ has $n$ columns, each having $k$ components. Thus, we can rewrite:

$$E = \{e_1, e_2, \ldots, e_n\} \quad (4.3)$$

where each $e_j = \{e_{j1}, e_{j2}, \ldots, e_{jk}\}$ implies episode $j$ represented as a $k$ component vector in $k$-dimensional space.

In the 2D and 3D cases, i.e. for $k = 2$ and 3, it is helpful to visualize the dimensionality reduction in the form of graphs as they provide interesting insights into the geometric “meaning” of the method. When $k = 2$, this implies, that the first column of $U$ is multiplied with the first singular value $s_1$, and the second column of $U$ is multiplied with the second singular value $s_2$, to get coordinates $\{o_{11}, o_{21}\}$ for the $m$ variables or parameters for
representing in a 2D plane, \( i = 1 \) to \( m \). Similarly, if \( s_1 \) and \( s_2 \) are multiplied respectively with the first and second columns of \( V \), we will get coordinates \((e_{1i}, e_{2i})\) for the \( n \) functions for representing in a 2D plane, \( j = 1 \) to \( n \). This 2D representation gives a visual idea of how the variables, parameters and objective and constraint functions are related to each other in the reduced \( k=2 \) space. In dimensional spaces greater than 3, the relationships cannot be directly visualized, but the same computations will be relevant. Figure 4.5 shows the 2D distributed graph representation for the example. Each of the events \((x_1, \ldots, x_6)\) and episodes \((h_1, \ldots, h_8)\) are plotted as points in this 2D space. Note how the episodes that share common events fall close to each other, while those that do not fall distant from each other.

### 4.5 Step IV: Similarity measurement

The final step involves ‘querying’ the \( k \)-reduced approximation space to infer the similarity patterns between events and episodes that becomes the basis for problem reformulation.

Because all the variables, parameters and functions are vectors in real space, and the position and distance between these vectors is a measure of the strength of the associative relationship between them (due to the SVD), “distance” measurements can be used as a measure of semantic relationship between any two elements. For example, if cosine measurements are used for measuring relationships, then two element vectors that have high cosine measurements will mean that they are semantically positively related. In other words, the distance measurement computes how semantically “similar” or “close” two design concepts are.

![Figure 4.5: Graphical representation of \( k \)-reduced approximation, \( k=2 \)](image-url)
Events that occur together in design experiences are expressed as symbols and are related together. Events that share semantic (structural or behavioral) relationships will tend to appear together in same context or episode (functions). Events that do not share semantic relationships will not tend to appear together in the same context or episode. Therefore, the higher the cosine measurement between two concepts (or “belongingness” of a concept to a cluster), the more semantic relation the concepts share, and vice versa.

REIFORM performs unsupervised similarity measurement based on two basic mathematical tools: 1) cosine similarity measurements between variables, parameters and functions signifying inference of “semantic distance” between variables, parameters and functions in the \( k \)-reduced space; and, 2) K-means clustering on \( k \)-reduced space to infer “semantically related groups” of variables, parameters and functions.

### 4.5.1 Cosine similarity measurement

Each point in Figure 4.5 is a vector in 2D space that represents a design variable, parameter or function. More generally, after the \( k \) reduction, each event and episode will be a point in \( \mathbb{R}^k \). A cosine angle measurement between any two vectors is a measure of semantic similarity – the higher the cosine angle value, the more the similarity. A cosine between any two vectors \( \mathbf{x} \) and \( \mathbf{y} \) is given by:

\[
\cos(\theta(\mathbf{x}, \mathbf{y})) = \frac{\mathbf{x}^T \mathbf{y}}{||\mathbf{x}|| \ ||\mathbf{y}||}
\]  

(4.4)

where \( \theta \) is the angle between \( \mathbf{x} \) and \( \mathbf{y} \). To show how the cosine measurement measures similarity and captures explicit as well as implicit associative relationships between events and episodes across varying \( k \) values, see Table 4-3. From the table, to continue with the same example, in the occurrence matrix \( A \) there is a 1 between \( x_1 \) and \( h_1; f_1 = x_1 + \exp(x_1 x_4) \) (explicit relationship), but a 0 between \( x_1 \) and \( f = f_1 + f_2 \) (explicit relationship), because \( x_1 \) does not directly occur in \( f \). It is evident, algebraically, that there is an implied relationship between \( x_1 \) and \( f \). In terms of cosine calculations, at \( k=2 \), a cosine measurement between \( x_1(1.5611, -0.93542) \) and \( h_1(0.9023, -1.3094) \) comes out to be 0.9100 (explicit relationship), which is very high as expected (they share a 1 in the original matrix). However, note that the cosine measurement between \( x_1(1.5611, -0.93542) \) and \( f(0.3253, -0.12589) \) comes out to be 0.9855 (implicit relationship), which is very high considering that they share a 0 relationship in the design occurrence matrix \( A \). This shows that the method brings out implied relationships existing between design elements and functions.

The original matrix has rank 8. Thus, setting \( k=8 \) should not return any implicit relationships but only the explicit relationships from the original occurrence matrix, because, as the degree of approximation increases, the values in the matrix will approach the original values in the occurrence matrix (refer to discussion in Sections 4.3.2 and 4.3.4). The matrix values should go to 0 and 1 as in the original occurrence matrix, and the cosines should reflect
this. As Table 4-3 shows, at $k = 8$, the cosine measurement between $x_1$ and $h_1$ is high (explicit relationship) at 0.7721, while the cosine measurement between $x_1$ and $h_1$ is low at 0.0684 (implied relationship).

<table>
<thead>
<tr>
<th>Occurrence Matrix A</th>
<th>$f_1 = x_1 + \exp(x_1) \cdot x_4$</th>
<th>$f = f_1 + f_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Occurrence Matrix Value, $k = 8$</td>
<td>$x_1$</td>
<td>1</td>
</tr>
<tr>
<td>Cosine value, $k = 8$</td>
<td>$x_1$</td>
<td>0.7721</td>
</tr>
<tr>
<td>Occurrence Matrix Value, $k = 2$</td>
<td>$x_1$</td>
<td>1.0204</td>
</tr>
<tr>
<td>Cosine value, $k = 2$</td>
<td>$x_1$</td>
<td>0.9100</td>
</tr>
</tbody>
</table>

Table 4-3: Cosine measurement variation at $k = \text{minimum and } k = \text{maximum values for } x_1$

### 4.5.1.1 Fixing a cosine threshold for returning semantically related events and episodes

To return a set of semantically related events and episodes in the $k$-reduced spaces, a cosine threshold is fixed. At each $k$ approximation, a query for measuring semantic similarity or closeness in this space implies that one point (design element or function) is chosen, and all other elements and functions that lie within this fixed cosine threshold are returned as answers. There is no absolute choice for fixing a cosine threshold. It is a matter of experimentation and varies with problem type. Also, the numerical values of the cosines are not a direct measure of relevance, rather the relative “intensity” ordering or gradation that they suggest is what should be used for deciding a threshold. For example, at $k = 2$, the values go as high as 0.9855 and 0.9100, but their relative difference is only 0.0755. For $k = 8$, the actual values are 0.7721 and 0.0684, but their relative difference is 0.7037.

Generally, a higher cosine threshold implies a stricter limit on similarity definition, and returns a fewer number of answers. A lower cosine threshold relaxes the limit and returns a higher number of answers. Similar to the interpretation in which $k$ is varied, varying a cosine threshold is also like varying “design freedom” which is an important concept in design modeling – a lower cosine threshold implies more freedom, as more number of related events and episodes will be returned. By varying cosine thresholds and observing different groups returned as answers, the designer can observe different possible formulations by increasing (low cosine threshold) or decreasing (high cosine threshold) “design freedom”. Chapter 8 will present effects of varying the cosine threshold to observe multiplicity in design reformulation decisions, and heuristics for choosing “good” cosine thresholds.

A query in this space can measure three types of similarity in terms of cosine distances: (1) event – event similarity (variables, parameters, design components); (2) episode – episode similarity (functions); and, (3) event-episode similarity.
4.5.1.2 Formal definition of interaction strength between events and episodes

To summarize, distance is the measure of similarity between events and episodes, where similarity implies the associative (explicit or implied) interaction strength between events and episodes. Thus, similarity is a measure of event-event interaction, episode-episode interaction, and event-episode interaction. Normally, “coupling” is understood as a binary relationship – there either exists, or does not exist a coupling between two variables or between a variable and a function. In REIFORM, though, coupling or interaction between two events and episodes can have variable strength or intensity. The definition for interaction strength between events and episodes is introduced.

Step II demonstrates that the action of SVD causes direct, explicit couplings (co-occurrences) between events and episodes in the occurrence matrix to be re-represented in real space in which distance is the measure of association strength in the \( k = r \) space. Step III demonstrates that the action of dimensionality reduction causes these associations to take on different intensities or strengths in different \( k \) reduced US and SV spaces. Step IV demonstrates that these intensities or strengths are measured in terms of a distance metric. Therefore, together these steps demonstrate that distance is a measure of the interaction strength between events and episodes, where interaction strength arises from the explicit as well as implied relationships between them. In this thesis, the cosine between two vectors was chosen as the distance metric. Thus, the cosine between two vectors is a measure of the interaction strength of their occurrence in the original problem representation.

Formally, interaction strength \( C_O \) between event \( i \) and event \( j \) is defined as the cosine between the vectors representing event \( i \) and event \( j \) in \( k \)-dimensional space \( \mathbb{R}^k \). Using Equation 4.2,

\[
C_O = \cos(\theta(o_i, o_j)) = \frac{o_i^T o_j}{||o_i|| ||o_j||} \tag{4.5}
\]

where \( o_i \) and \( o_j \) are \( k \) component vectors in \( k \)-dimensional space, and \( k = 1 \) to \( r \).

Interaction strength \( C_E \) between episode \( i \) and episode \( j \) is defined as the cosine between the vectors representing episode \( i \) and episode \( j \) in \( k \)-dimensional space \( \mathbb{R}^k \). Using Equation 4.3,

\[
C_E = \cos(\theta(e_i, e_j)) = \frac{e_i^T e_j}{||e_i|| ||e_j||} \tag{4.6}
\]

Interaction strength \( C_{OE} \) between event \( i \) and episode \( j \) is defined as the cosine between the vectors representing event \( i \) and episode \( j \) in \( k \)-dimensional space \( \mathbb{R}^k \). Using equations 4.2 and 4.3,

\[
C_{OE} = \cos(\theta(o_i, e_j)) = \frac{o_i^T e_j}{||o_i|| ||e_j||} \tag{4.7}
\]

Further, let \( X(k) \) \((m \times m)\) be a matrix that contains all the interaction strength measurements \( C_O \). \( X(k)_{ij} \) is the interaction strength between event \( i \) and event \( j \) in \( \mathbb{R}^k \). Let \( Y(k) \) \((n \times n)\) be a matrix that contains all the interaction strength measurements \( C_E \). \( Y(k)_{ij} \) is the
interaction strength between episode $i$ and episode $j$ in $\mathbb{R}^k$. Let $Z(k) (m \times n)$ be a matrix that contains all the interaction strength measurements between event $i$ and episode $j$. $Z(k)_{ij}$ is the interaction strength between event $i$ and episode $j$.

Chapters 5, 6 and 7 will demonstrate how such similarity measurements assist with design reformulation tasks. The cosine threshold approach is extended into a matrix partitioning approach in Chapter 5 in order to use REIFORM for design decomposition type reformulation tasks.

### 4.5.2 Unsupervised K-means clustering

(Note: For differentiating between the number of retained singular values in the dimensionality reduction step, the notation used is a small, italicized $k$. For K-means clustering, the notation used is a capital $K$.)

For some reformulation tasks, highly associated sets of events and episodes need to be clustered into groups. One example is design decomposition, where a large problem is to be decomposed into smaller sub-problems with local variables and functions, and linked to the main problem by linking variables and sub-problems.

This thesis presents (Chapter 5) two equivalent methods for performing such decomposition. The cosine threshold approach as described above is extended into a matrix partitioning approach, described in Chapter 5. However, it is always beneficial to have a second equivalent method validate the results produce by a first one. The other approach is to use an unsupervised clustering approach. Results from both methods could then be compared for equivalence or difference.

In this work, a K-means clustering algorithm is used. The K-means clustering algorithm is an iterative method for putting $N$ data points in an $I$-dimensional space into $K$ clusters, where each cluster is parameterized by a vector $\mathbf{m}(k)$ called its mean (Mackay, 2003). The data points are vectors denoted by $\mathbf{x}(n)$, where $n = 1:N$, each $\mathbf{x}$ has $I$ components $x_i$. Each data point is assigned to the nearest mean (or centroid). A metric is defined for measuring distances between these data points (e.g. cosine, Euclidean, city-block, etc.). For this work, we have used a cosine distance between data points. In an update step, the means are adjusted to match the sample means of the data points within a cluster. This algorithm always converges to a set of clusters, but initializing the means at different points can produce different clusters for the same data set.

K-means clustering can be used for clustering the $N$ data points in the dimensionally reduced space, representing design variables and functions (or just design components in the non-analytic case).

Both the cosine similarity measurements and the K-means clustering, in effect, represent the same aim – producing groups of semantically related variables and functions. The only
difference is in the process structure: in cosine measurements, a cosine threshold value has to be chosen to observe the groups, while in K-means clustering, groups are returned on the basis of what distance function (e.g. cosine, Euclidean, city-block etc.) is chosen to calculate group membership. In this thesis, when we choose “cosine” as the distance metric type for K-means clustering, the results produced by both algorithms are equivalent. Having two different methods provide the same results demonstrates the robustness of the method. It also evaluates whether using two different clustering algorithms returns similar answers – this serves as a test to confirm that the method is returning consistent answers in semantic terms.

4.6 A note on computational implementation

REIFORM was implemented in the form of a set of functions and programs in MATLAB Version 7 R14. MATLAB has inbuilt functions for computing both the SVD and K-means clustering. Computation time for the full method will be dominated by the SVD and K-means clustering algorithms. In practice, fast and efficient algorithms exist for computing both the SVD and K-means. MATLAB uses LAPACK (Anderson, 1999). In design modeling, the largest matrices would go to thousands of rows and columns. The time complexity calculations only become important for matrices of dimension > $10^5$. REIFORM has not been tested on very large scale problems, although it is known that SVD is stable for very large matrices.

4.7 Summary

This chapter presented the method REIFORM that acquires semantic knowledge from the syntax of design representation. The method has 4 main parts: (1) generating a common representational form – the occurrence matrix $A$ from analytical and non-analytical design formulations; (2) performing SVD of the matrix giving $A = USV^T$; (3) performing dimensionality reduction to produce an approximation of $A$; and (4) computing similarity measurements between design variables and functions in the reduced dimension space using cosine measurements and unsupervised K-means clustering.

REIFORM calculates linear approximations of the associative patterns of symbol co-occurrences in a design problem representation to infer induced interaction/coupling strengths between variables, constraints and/or system components. Unsupervised clustering of these approximations is used to identify useful reformulations. These two components of the method automate a range of reformulation tasks that have traditionally required different solution algorithms. It is not dependent on knowledge-based rules and processes but on viewing problem reformulation as a pattern recognition problem. Therefore, it can be applied
to any problem reformulation tasks that require variables and functions to be chosen or grouped according to their semantic-symbolic relationships with each other, without the need to provide it with any domain specific semantic knowledge. The range of problem reformulation tasks chosen for demonstration in the thesis may not be exhaustive, but they cover a broad spectrum to demonstrate this assertion. A variety of design domains and problem sizes, represented using some principal mathematical forms, were chosen. The next three chapters demonstrate how this method can be used to perform different types of reformulation tasks for analytically and non-analytically formulated problems.

Because SVD is known to be stable as an algorithm, this approach seems particularly relevant for problem formulation decisions for medium to large scale problems, where the number of design variables, parameters and constraints are too large for choices to be determined by direct human observation of design relationships, or where existing methods need much computational effort.
Chapter 5

Design Decomposition and Modularity Analysis

By object is meant some element in the complex whole that is defined in abstraction from the whole of which it is a distinction.

John Dewey, Qualitative Thought in Philosophy and Civilization

This chapter focuses on the development and application of REIFORM for two types of problem reformulation tasks – (i) design decomposition; and (2) modularity analysis for complex systems. Additionally, the performance of REIFORM is empirically validated and demonstrated by testing its ability to return “correct” sub-problems for problem decomposition type tasks using a problem where a correct formulation and decomposition into sub-problems is pre-defined. Both FDT and DSM representation based problems are used for example demonstrations. Complex systems may not, often, lead to one exact solution – one aim in this chapter is to show how, using the different dimensional reductions and cosine thresholds, multiple useful interpretations or reformulations can be inferred. As Dewey’s quote shows, defining an object as a component of a system is an act of abstraction. The abstraction of a sub-system from a system, therefore, will always be informed by multiple perspectives. Understanding the range of varying interaction strengths between system components or variables is one such key perspective that can lead to multiple solutions.

5.1 Design decomposition

Decomposition of design problems is an important part of formulation and reformulation performed at the conceptual design stage. Decomposition decisions taken at the pre-modeling stage can significantly affect the way in which design problems are modeled as components within a larger system or as a full system. Complex product design is typically characterized by either large problem sizes in terms of the number of design variables and constraints, or
strong interactions between them, usually defined by analytical functional constraints (Michelena & Papalambros, 1997), or logical dependencies between design elements (Pimmler & Eppinger, 1994) or any other kind of implicit design relationship (results of simulation etc.). Whether the complexity stems from a large problem size or strong interactions, solving such models for a feasible result is difficult and frequently involves the application of numerical algorithms onto the problem model. Even if a problem formulation is not complex, it is good design practice to decompose it into smaller sub-problems if the problem structure is “decomposable”.

Among other benefits, decomposition can reduce computational effort to solve a design problem, and enable coordination and concurrency in design development. Often, large model sizes reduce the reliability and speed of numerical solution algorithms (Michelena & Papalambros, 1997), making a solution process difficult. Decomposing a problem can mean that different solution algorithms can be chosen for the different sub-problems. Apart from these computational benefits, one of the main ideas inherent in decomposition is its use for conceptual design – referring to Dewey’s quote again, conceiving of a system in terms of sub-systems (or an analytically formulated problem in terms of sub-problems) before and while a detailed analytical-mathematical model is developed and finalized. Once an initial formulation has been derived, decomposing the problem using multiple perspectives and in multiple ways can aid the development of a “good” formulation. Such an analysis may reduce problem complexity (dimensionality and interactions) and aid a designer to understand and simplify the system.

5.1.1 Existing research

In general, a decomposition method partitions the original master problem into sub-problems, and implements an appropriate coordination strategy (Michelena & Papalambros, 1997; Wagner & Papalambros, 1993a). The smaller problems are solved independently but are coordinated by a master problem. Local variables and functions define the sub-problems, while linking variables and functions define the coordination strategy. Linking variables are those variables that affect independent sub-problems when held fixed. Linking functions are those functions that affect independent sub-problems when deleted or relaxed. Therefore, if the linking variables and functions are removed, then this can reveal independent sub-problems. Prevalently, matrix and graph representations have been employed to solve decomposition problems. For extensive reviews, refer to Michelena (1997) and Wagner (1993a)).

Decomposition approaches have been classified in various ways – object-based, aspect-based or sequential (Michelena & Papalambros, 1997) or product, problem and process based (Kusiak & Larson, 1995). The general conceptual approach presented as the basis of most
decomposition methods presented in the literature, e.g., (Li & Li, 2005; Michelena & Papalambros, 1997; Pimmner & Eppinger, 1994; Sosa et al., 2003; Wagner & Papalambros, 1993a) corresponds to Simon’s definition (Simon, 1969/1981) of a “nearly decomposable system” – a system that is characterized by weakly interacting sub-systems with strong local interactions within each sub-system.

The research literature is rich with many exact and optimal decomposition methods. Therefore, the primary motivation for developing and applying REIFORM for design decomposition was not to produce a “better” algorithm that can produce optimal decompositions in lesser time than established methods. Rather, the main motivation was to explore the role of Singular Value Decomposition and unsupervised clustering approaches to design decomposition because of their capacity to reveal global implicit information using local explicit information. It appeared that the analysis and argument developed in Chapter 4 resonated with the approach in Simon’s definition – the idea of abstracting sub-systems based on mutual interactions between components.

There is a class of decomposition algorithms that pose the problem in terms of a graph based representation and use spectral methods to solve the decomposition problem (Michelena (1997) describe such an approach, as well as include many references to other similar work). In such approaches, the eigenvalue-eigenvector structure of the adjacency matrix of the graph is employed. More specifically, for partitioning the problem representation graph into P partitions, a P-dimensional geometric representation of the graph is constructed using the P eigenvectors that correspond to the smallest eigenvalues of the graph Laplacian matrix to arrive at optimal decomposition results. Although mathematically rigorous, this approach is not conceptually simple. For example, all graph algorithms by necessity require a square matrix representation. The design problem has to be re-represented as a graph or a hypergraph to enable a graph-based analysis.

In contrast, as shown in Chapter 4, problem representation and conceptual interpretation used in REIFORM is different and simpler than these approaches. The approach presented in REIFORM is inspired by the way SVD and dimensionality reduction are used in other knowledge domains (specifically statistical natural language processing). This approach offers a conceptual simplicity that seems to be missing in the other approaches. For example, because SVD is itself a special “best” known decomposition for any general rectangular matrix, there was no need to overlay a graph or hypergraph representation. The design representation matrix itself could be analyzed. Secondly, SVD coupled with the dimensionality reduction and unsupervised similarity measurement approach could be interpreted in terms of producing a continuous, distance based, generalized clustering starting from direct interactions between variables, functions and system components. This approach therefore seemed deserving of a closer analysis. Because the prevalent decomposition
algorithms, for example the graph-spectral methods based algorithms, are usually complex computational constructions and require much development effort on part of the designer, the simplicity of the method structure as suggested by REIFORM was particularly inspiring. The results of such a preliminary method development suggest that this approach can be further pursued to develop an exact and optimal decomposition method. Ideas on such theoretical method development and formal proof are included as future work in Chapter 10.

5.1.2 Using REIFORM for decomposition

The main conceptual idea behind using REIFORM for design decomposition is that it converts interaction strengths between variables and functions into a continuous distance based representation, i.e. each variable or function is plotted in an $r$ or $k$ dimensional space as a vector, and the (cosine) distance is a measure of the interaction strength between them.

To summarize the analysis from Chapter 4, the $UV$ and $SV^T$ spaces show that any linear combination of this special form is a valid approximation of the original space $A$ because the orthonormal vectors and the singular values are produced by using that very space. Approximations to these spaces imply using different linear combinations, that is, different numbers of singular values. It is known that in decreasing order of magnitude, the singular values capture the most important associative patterns. Therefore, the approximations, that are linear least square approximations, will overplay the most important associative patterns and underplay the rarer ones. That is, variables and functions that mutually share high relations will be strengthened in the earlier approximations, and those that do not will be weakened.

This is exactly the characteristic we need for design decomposition if we were to follow Simon's definition. In design decomposition, (specific case as well as general case as used in the thesis), we need to identify groups of variables and functions that share strong interactions with each other, as separate from those that do not share strong interactions with this group. We need to do this for all the groups that may be present in the single data set, simultaneously. This is a min-max problem - maximize intra-sub-problem interaction and minimize inter-sub-problem interaction. In a generalized, continuous way, the SVD and dimensionality reduction does just this – At some $k$ values, this step intuitively minimizes the distance between strongly connected variables and functions and pulls them closer in this re-representation space due to the $k$ largest singular values that capture the strongest relations. At the same time, it maximizes the distance between the not strongly connected variables and functions and pushes them far in space, again due to disregarding the $(r-k)$ smallest singular values treating the weaker association patterns as “noise”.

Therefore, each approximation is a valid approximation to identify these groups. At each $k$ level, we aim for this min-max arrangement as the preferred decomposition pattern and interpret the patterns that prevent this from being identified as “noise”. Therefore, even
though all approximations are valid approximations of the original relational matrix, i.e. valid approximations of the design problem representation, we need to identify only those approximations that lead to this min-max cut. In other words, we need to identify those \( k \) values that lead to a well-formed (well-decomposed) problem in which the distances between those variables and functions that share strong mutual interaction is minimized (interaction is maximized) and distances between those variables and functions that do not share strong mutual interactions in maximized (interaction is minimized). Thus, some \( k \) values will lead to well-decomposed problems in terms of identifying the sub-problem clusters. To find out which \( k \) values lead to such well-decomposed problems, a matrix partitioning and unsupervised clustering approach is developed in this chapter. The choice of the \( k \) value is crucial.

A limitation of the method in this current form, is that the \( k \) value or values that leads to the optimal decomposition can only be heuristically identified. Chapter 8 presents these heuristics to be applied along with the partitioning and clustering approaches presented in this chapter to correctly identify the range of \( k \) values at which well-formed reformulations are returned. Chapter 10 discusses possibilities for an exact and optimal decomposition method. However, since there is a way to limit the search for implicit information within a range of \( k \) values, and decomposition is not possible on only explicit information in the original occurrence matrix, this does not present a serious limitation. One can see a full range of multiple well-decomposed solutions by varying the \( k \) values for the same design problem representation. If there is a solution, or solutions, the method is able to identify these.

5.1.3 Method for decomposition

1. Create the occurrence matrix (Step I of REIFORM) from an analytical or non-analytical formulation of any design problem. In the analytical case, the occurrence matrix is generally rectangular, similar to the FDT. In the non-analytical cases, the occurrence matrix can be either rectangular (as in cases where response surface or simulation results form the FDT) or square (DSM representations). The occurrence matrix contains the explicit local interaction information between design concepts (variables, functions, design components etc).

2. Use the SVD analysis on the occurrence matrix (Step II of REIFORM) to convert the discrete (in many cases binary, but not necessarily so) local interaction information into a continuous distance based representation. That is, each event or episode (variable, function, design component) is represented as a point or vectors in \( r \) dimensional space (rank of occurrence matrix), and the distance between two events and episodes is a measure of the interaction strength between them. This captures global information about problem structure as inference can now be made for two
design concepts that are not explicitly and locally related in the occurrence matrix representation.

3. Use the dimensionality reduction (Step III of REIFORM) to observe different induced patterns of interaction between the design concepts in reduced $k$-dimensional space. As Chapter 4 demonstrated, the first few singular values preserve the strongest explicit and implicit patterns, and can reveal multiple possible interaction patterns as combinations of the linearly independent bases. This leads to multiple reformulations. Some $k$ approximations lead to identification of well decomposed problems following the argument presented in the previous section.

4. To identify the decompositions, compute distances between design concepts in the $k$ reduced space using a cosine distance function and produce cosine similarity matrices ($X(k)$, $Y(k)$ and $Z(k)$) (Step IV of REIFORM). Start with $k=2$.

5. If a well-formed decomposition exists at this $k$ level, this is identified through a matrix partitioning approach using a cosine threshold on these matrices. A greedy algorithm was developed in this thesis for this purpose, to be presented in Section 5.1.4, but any established matrix partitioning algorithm can be employed for this purpose. If no well-formed solution is returned, then use the next $k$ value. If a well-formed solution is returned, but the designer wishes to explore other solutions, then use the next $k$ value.

6. Equivalently, identify sub-problems through an unsupervised clustering approach using K-means algorithm on the vector representations of events and episodes in the $k$-reduced space directly (Step IV of REIFORM).

Steps 4, 5 and 6 require performing the analysis over a range of $k$ values, and steps 4 and 5 require choosing a cosine threshold – these are parametric choices. Chapter 8 presents heuristics for choosing these by employing parametric studies. Since the first 4 steps and step 6 have been described in Chapter 4, the next two sections describe Step 5 and a small description of step 6 specifically with reference to design decomposition. Finally, these steps are demonstrated using two examples – the FDT based example used in Chapter 4 (Michelena & Papalambros, 1997) and a DSM based example (Pimmler & Eppinger, 1994).

5.1.4 Matrix partitioning approach for decomposition

When the occurrence matrix derives from an FDT, recall that $X(k)$, $Y(k)$ and $Z(k)$ were defined as the variable-variable, function-function and variable-function cosine similarity matrices in $k$-reduced space. When the occurrence matrix derives from a DSM, for a matrix with $m$ rows (design components), ($i = 1$ to $m$) and $n$ columns (design components), ($j = 1$ to $n$), $m=n$, $X(k)$ is the component $i$ – component $j$ cosine matrix, $Y(k)$ is the component $j$ –
component \( i \) cosine matrix and \( Z(k) \) is the combined \((i-j)(j-i)\) cosine matrix. Where such a matrix is symmetric, all these three reduce into one matrix.

### 5.1.4.1 Matrix partitioning definition:

A mathematical model of a design problem represented as the occurrence matrix \( A \) is decomposable into \( F \) sub-problems if there exist \( F \) block matrices in the event-episode coupling strength matrix \( Z(k), Z_1(k), Z_2(k), \ldots Z_F(k) \), such that:

(i) the entries of each \( Z_i(k), 1 \leq i \leq F \), are higher than the cosine threshold being considered;

(ii) if there are other values in the matrix \( Z(k) \) (not a part of these \( F \) block matrices) that are higher than the cosine threshold, then they do not form a defined block matrix of a size comparable to these \( F \) block matrices.

Then, the events and / or episodes that are represented as the rows and columns of each block matrix \( Z_i(k) \) is a sub-problem of the main problem \( A \). The residual matrix \( Z(k) - Z_1(k) - Z_2(k) - \ldots Z_F(k) \) represents the co-ordination chunk of the linking events and episodes.

Note in this definition that there are two parameters – the cosine threshold value and the dimensionality reduction \( k \) value. These parameters are chosen by the user. Changing these parameters may lead to different decompositions. If a solution exists at a \( k \) level, then the algorithm will identify it. The designer is required to reapply the steps for a range of \( k \) levels.

To identify which range of \( k \) values (those that return implicit information), Chapter 8 presents parametric studies and heuristics. Intuitively, the “best” system decomposition would be one that results in disjoint block matrices of roughly similar sizes, with no overlapping common elements, and no remaining linking variables or function chunks. That is, if cast in optimization terms, a cosine threshold and \( k \) value is to be chosen to (i) identify balanced (similar sized) block matrices and (ii) minimize the co-ordination chunk that comprises the linking variables and functions.

### 5.1.4.2 Algorithm to identify the block matrices

A greedy algorithm was developed to identify these block matrices. The algorithm takes in the cosine distance matrices \( X(k), Y(k) \) and \( Z(k) \) for a certain \( k \) value and a certain cosine threshold and returns block matrices in terms of reordered rows and columns. The algorithm (Figure 5.1) reorders rows and columns of \( X(k) \) and \( Y(k) \) in terms of decreasing similarity with each other, and this ordering is then used to reorder the \( Z(k) \) matrix rows and columns. This reveals the block matrices.
5.1.5 Unsupervised K-means clustering approach for decomposition

If instead of a matrix partitioning approach a clustering approach is adopted, this facilitates a similar sub-problem identification. As discussed in Chapter 4, K-means clustering is an unsupervised concept clustering algorithm that chooses a cluster configuration in which the distance between the concepts (normally represented as points in an n-dimensional space) in a cluster is minimized. Both partitioning and clustering approaches were chosen to validate whether they return similar solutions.

5.1.6 FDT example demonstration

This section now presents method demonstration and results using the FDT example derived from the problem in (Michelena & Papalambros, 1997) (Chapter 4). The first 4 steps described in Section 5.1.2 have already been described in Chapter 4 (Figures 4.1 – 4.5). Steps 5 and 6 are demonstrated here.

5.1.6.1 Matrix partitioning using a cosine threshold

The example problem used in the last chapter (Michelena & Papalambros, 1997) contained 8 events (6 design variables, and the symbols \( f_1 \) and \( f_2 \)) and 9 episodes (1 objective function and 8 constraints). Steps I, II, III and IV of REIFORM are applied onto this problem, and fixing \( k = 2 \), \( X(2) \), \( Y(2) \) and \( Z(2) \) matrices are computed. Considering a cosine threshold of 0.7, block matrices are identified using the algorithm described in Figure 5.1 (Matlab code for the algorithm is enclosed separately in the CD, Refer to Appendix A). Figure 5.2 shows the decomposition results in the \( Z(2) \) matrix. Note how the rows and columns have been reordered and the values in red show the identified block matrices.
Two linking variables were identified—\( x_1 \) and \( x_3 \). Two block matrices were identified: Sub-problem 1 with variables \( \{ x_4, x_6 \} \) and functions \( \{ h_1, h_6, h_8, h_5 \} \); and sub-problem 2 with variables \( \{ x_2, x_5 \} \) and functions \( \{ h_4, h_3, h_7, h_2 \} \). This result matches with the decomposition suggested by Michelena and Papalambros (1997). The choice of the cosine threshold and \( k \) is important. Heuristics will be presented in Chapter 8 for choosing “good” cosine thresholds and \( k \) values that return well-formulated decomposition results.

### 5.1.6.2 Clustering using K-means algorithm

If a clustering approach is adopted instead of the above approach, the K-means algorithm equivalently identifies these independent sub-problems. The K-means algorithm was applied to the event and episode vectors, \( \mathbf{O} \) and \( \mathbf{E} \) (Section 4.4), in the \( k = 2 \) reduced space, with the distance metric type set to “cosine” onto the example problem. It may be noted here that setting another distance metric type such as Euclidean may alter the clustering suggested. For this case, the cosine type seemed the most relevant because a cosine measurement captures both the distance and direction effects (as opposed to Euclidean which will capture only distance in Euclidean space). Figure 5.3 shows the results of K-means clustering superimposed onto the 2D graph representation (Figure 4.5) for visual interpretation.
The results almost exactly match with the decomposition suggested by the matrix partitioning approach of Michelena and Papalambros (1997): sub-problem 1 with local variables \{x_2, x_5\} and functions \{h_2, h_3, h_7\}, and sub-problem 2 with local variables \{x_4, x_6\} and functions \{h_1, h_5, h_6, h_8\} and \{x_1, x_3\} as linking variables. In this result, the only difference is that the K-means algorithm identifies \(h_4\) as part of the cluster with the linking variables, i.e. as a linking constraint. This is explained by the observation that constraint \(h_4\) has two variables that are linking variables, and only one local variable, so the graph shows that it has high cosines with both the \{x_2, x_5\} cluster as well as the linking variables chunk.

**5.1.6.3 No partitioning or clustering is possible at full rank, \(k=8\)**

Figure 5.4 shows the original occurrence matrix, and the cosine measurement matrix \(Z(8)\). At \(k = 8\), the approximation matrix \(A'\) becomes equal to the occurrence matrix \(A\) as all the singular values are used to produce the approximation. Thus, the cosine distances between variables and functions will correspond to distances in \(k = r\) dimensional space. Note how the cosine distance matrix \(Z(8)\) shows higher than the threshold value (0.7) only in the positions corresponding to 1 values in the occurrence matrix.

<table>
<thead>
<tr>
<th>(A) (Occurrence Matrix)</th>
<th>(h_1)</th>
<th>(h_2)</th>
<th>(h_3)</th>
<th>(h_4)</th>
<th>(h_5)</th>
<th>(h_6)</th>
<th>(h_7)</th>
<th>(h_8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_1)</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(x_2)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>(x_3)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(x_4)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(x_5)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(x_6)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(k=8), cosine threshold = 0.7</th>
<th>(h_1)</th>
<th>(h_2)</th>
<th>(h_3)</th>
<th>(h_4)</th>
<th>(h_5)</th>
<th>(h_6)</th>
<th>(h_7)</th>
<th>(h_8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_1)</td>
<td>0.7721</td>
<td>0.1794</td>
<td>0.7194</td>
<td>0.7545</td>
<td>0.1762</td>
<td>0.8397</td>
<td>0.2572</td>
<td>0.0720</td>
</tr>
<tr>
<td>(x_2)</td>
<td>0.1118</td>
<td>0.8220</td>
<td>0.8945</td>
<td>0.7185</td>
<td>-0.0385</td>
<td>0.1639</td>
<td>0.9070</td>
<td>0.1087</td>
</tr>
<tr>
<td>(x_3)</td>
<td>0.0770</td>
<td>0.0735</td>
<td>0.2112</td>
<td>0.7813</td>
<td>0.1567</td>
<td>0.0989</td>
<td>0.0029</td>
<td>0.8412</td>
</tr>
<tr>
<td>(x_4)</td>
<td>0.8151</td>
<td>-0.0363</td>
<td>0.1757</td>
<td>0.1410</td>
<td>0.7908</td>
<td>0.8788</td>
<td>-0.0404</td>
<td>0.1131</td>
</tr>
<tr>
<td>(x_5)</td>
<td>0.0431</td>
<td>0.8776</td>
<td>0.8472</td>
<td>0.3288</td>
<td>0.0103</td>
<td>0.0673</td>
<td>0.9583</td>
<td>-0.0435</td>
</tr>
<tr>
<td>(x_6)</td>
<td>0.1028</td>
<td>0.0065</td>
<td>-0.0459</td>
<td>0.1143</td>
<td>0.8397</td>
<td>0.1414</td>
<td>0.0077</td>
<td>0.8579</td>
</tr>
</tbody>
</table>

**Figure 5.4: No decomposition is possible with only full-rank explicit information at \(k = 8\)**

Applying the partitioning algorithm on this matrix fails to produce the decomposition solution. The reordering algorithms cannot show any reasonable sized block matrices. Similarly applying the K-means clustering on the vector representations of the events and episodes in the full rank \(r\) space will fail to identify the sub-problems.
5.1.7 DSM example demonstration

In the DSM form, the occurrence matrix captures a mapping between the same elements – event-event mapping, making it a square matrix. The mapping may capture symmetric design relationships (such as spatial adjacency) or asymmetric design relationships (such as material exchange or information flow). In this section, an automotive climate control system (Pimmler & Eppinger, 1994) is presented to demonstrate system decomposition.

Figure 2.5 shows the DSM representation for a Ford automotive climate control system (Pimmler & Eppinger, 1994). The system is to be decomposed component-wise into subsystems with four kinds of constraints (interaction types): spatial adjacency, energy interactions, materials interaction and information exchange between components. The interaction strengths range from -2 to +2, signifying negative as well as positive interactions. For example, because the radiator and engine fan are functionally coupled with each other, they get a +2 score for both spatial adjacency and material interaction conditions. For the occurrence matrix generated from the original DSM, refer to Matlab code enclosed in the CD. Appendix A provides the list of programs by example names.

The only difference between the DSM form and the occurrence matrix form used by REIFORM is that the DSM has undefined or zero diagonal entries, while REIFORM assumes the maximum value in the diagonal with the assumption that any design component has a maximum interaction with itself in terms of all the four interaction types considered. Each of the four interaction types (spatial, energy, information and material) impose different sets of constraints onto the design components and may result in conflicting decisions on how the components are to be decomposed as sub-groups. Also, these are likely to lead to different decomposition decisions that affect decisions like design team formation decisions or organizational boundaries. REIFORM is applied to each of the interaction matrices separately.

The authors in the source paper (Pimmler & Eppinger, 1994) do not describe in detail the clustering algorithm they have employed in reaching the results. They state that they have used a heuristic swapping algorithm that operates on the basic principle of re-ordering rows and columns such that the positive elements cluster round the main diagonal. The results from the two methods are compared.

5.1.7.1 Decomposition results from the “Material” interaction type matrix

This section presents the results of applying REIFORM to the “materials exchange” interaction matrix. The original data matrix has 16 system components. However, only 10 of these participate in the material interaction. The occurrence matrix is thus a $10 \times 10$ size matrix. In this particular case, the matrix entries are symmetric. This means that the $U$ and $V$ matrices, and, consequently, the coupling strength measurement matrices $X(k)$, $Y(k)$ and $Z(k)$
will be the same. Any of these three matrices can be used to perform the decomposition, as the matrix partitioning approach will collapse to the single step of reordering the rows and columns of the square matrix in a descending order. Preserving the first two singular values, i.e. \( k=2 \), produces all the relevant decomposed sub problems (Figure 5.5).

As solution 1, REIFORM identifies three main sub systems as a result of decomposition, considering 0.8 as the cosine threshold for membership within a particular sub system: (1) components A, B and E; (2) components E, F, I and H; and (3) components C, P, O and G. Figure 5.6 shows the results from the source paper (Pimmler & Eppinger, 1994). Thus, solution 1 has three sub-systems consisting of 3 components, 4 components and 4 components each, with one shared component. These results are almost identical to those reported in the source paper with the following differences. In REIFORM, component H is classified only in the second sub system, instead of being a shared part of the second and third sub systems as reported in (Pimmler & Eppinger, 1994). This happens because H has positive interactions with only component P in the last sub system and with no other element, thereby lowering its inclusiveness in the last sub system.

Further, if the cosine threshold is lowered to 0.5, an alternate decomposition, solution 2, is suggested: (1) components A, B, E, F, I, H and (2) components C, P, O, G. Components A and B show high cosine measurements with components F, I, H due to the fact that both these sets co-occur with E. Since the method works by measuring distributed patterns of associations between all elements it reveals implicit, indirect associations as well as explicit, direct ones. Even those elements that do not directly co-occur with each other show high semantic relationship with each other on the ground that they are related through co-occurrence with a third element. Solution two shows two sub-systems, one with 6 components, the other with 4. Comparing the two solutions, observe that solution 1 has one linking component, but similar sub-problem sizes, whereas solution 2 has no linking...
components, but sub-problem sizes are somewhat more unbalanced as compared to solution 1. It would appear that if one wanted to minimize the number of coupling components, one would prefer solution 2. However, if the objective was to aim for balanced size sub-problem clusters, one would select solution 1. Solution acceptability will depend upon which one is more cost-efficient when actually assessed for any optimization or design objective. The “multiple” outcome result is interesting because it shows that designers can “tune” the cosine threshold and number of clusters to observe multiple, well-formed decompositions. This is advantageous because it is frequently difficult to objectively select an “optimal” decomposition for complex design problems; sub-system boundaries can be decided on non-design related criteria such as organizational boundaries (Sosa et al., 2003).

Applying the K-means algorithm with cluster numbers set to 3 and 2 respectively produce the same two solutions.

![Figure 5.6: Results of decomposition from source paper (Pimmler & Eppinger, 1994)](image)

The authors in the original paper do not mention whether their solution is optimal, only that a heuristic algorithm has been used. Therefore, for specific optimization tasks, the solutions produced by REIFORM (solution 1 largely matches with the original paper solution, and REIFORM shows an additional solution 2) would have to be tested using a numerical optimizer. Note that changing just one interaction value in the occurrence matrix will produce changes in all the resulting matrices and cosine values, which may then alter the decomposition decision suggested. As observed above, this allows designers to experiment with the matrix entries, i.e. the interaction values in an efficient way to observe how the decomposition decision may change. For complex systems, the decision whether or not an interaction between two design components is to be represented or not, is itself a modeling decision. Similarly the interaction values may result from different considerations.

### 5.1.7.2 Effect of changing k values

To identify if there are any other well-formed solutions, the k levels were varied and the matrix reordering algorithm reapplied at increasing k levels. Figure 5.7 shows that at cosine threshold 0.5, solution 1 is also identified at k = 3.
Figure 5.7: Decomposition solution at \( k = 3 \), cosine threshold = 0.5

No solutions (well formed block matrices) were returned from \( k = 4 \) onwards by applying the matrix partitioning algorithm, as the values increasingly approached the limit set by the original occurrence matrix, rank = 9. Figure 5.8 confirms that at \( k = 9 \), no well-formed decomposition solution is returned. The top matrix shows the reordered \( Z(8) \) matrix; the matrix partitioning algorithm reorders the rows and columns based on the values, but no valid partitions can be identified using the cosine threshold of 0.5. The bottom matrix shows the \( A' \) approximation using all the singular values. As expected, this reproduces the original occurrence matrix. Note that all the positions with original '2' values correspond to the higher than cosine threshold values in the top matrix, the other values are close to '0'. This confirms that at full rank, \( k = 9 \), only the explicit relational information is retrieved and thus, in the absence of implicit information, no decomposition is obtained.

Figure 5.8: No decomposition solution is obtained at \( k = 9 \), full rank approximation
5.2 Modularity analysis for complex systems

In this section, REIFORM is applied to demonstrate the analysis of modular versus integrative systems in complex systems. Modularity analysis is a task similar in spirit to design decomposition. Whereas in design decomposition, a given system has to be decomposed into sub-systems, in modularity analysis, the sub-system decomposition is given and designers study which sub-systems are modular and which one integrative. Modular systems have been qualitatively defined (Sosa et al., 2003) as those systems whose cross boundary design interfaces are concentrated among a few other systems (usually spatially contiguous systems). On the other hand, integrative systems are defined as those systems whose design interfaces are scattered among components in (almost) all the systems that comprise the product. This qualitative definition is interpreted in quantitative terms for REIFORM as follows: modular and integrative systems form two ends of a spectrum and sub-systems lie on this spectrum as measured by the number of other sub-systems that a certain sub-system shares design interactions with.

The analysis of modularity becomes important especially in complex systems. Complex systems are identified by two characteristics – large problem size and high coupling between system components. In complex systems, it is rare for a system to be perfectly decomposable, i.e. it is rare for a complex system to be perfectly modular. One of the main problems is that it becomes difficult to identify subsystem boundaries because components within a sub-system share design interfaces with components from other sub-systems. It is for this reason that designers aim for a “nearly decomposable system” rather than a “perfectly decomposable one”. It becomes important to analyze which sub-systems within this large system are modular, which ones integrative, and to what degree.

REIFORM is applied for modularity analysis using the same approach as for decomposition. Steps I to IV are applied onto a design problem representation. For different $k$ values, decomposition analysis is performed using the matrix reordering algorithm and the K-means clustering algorithm. However, in this the objective is not decomposition (the sub-systems are already known) but the objective is to analyze inter-sub-system interactions. Measuring the inter-sub-system interactions allows the measurement of which systems are modular and which ones integrative and to what degree. Since the cosine measurements can be performed between any two components from any two sub-systems, the method will be able to correctly identify high interactions between components from different sub-systems and between components from the same sub-system. It is the former characteristic that helps to identify the modular and integrative systems.
### 5.2.1 Example demonstration

Modularity analysis is demonstrated on a large scale design problem. Figure 5.9 shows the DSM representation for a large Pratt and Whitney commercial aircraft engine (Rowles, 1999; Sosa et al., 2003) with 54 design components in 8 subsystems (Fan, Low Pressure Compressor (LPC), High Pressure Compressor (HPC), Combustion Chamber (CC), High Pressure Turbine (HPT), Low Pressure Turbine (LPT), Mechanical Components, and Externals and Controls. The aeroengine problem was chosen because it shows both characteristics of a complex system – large problem size and strong sub-system interactions.

![Design interaction matrix for the Pratt and Whitney aeroengine problem (Sosa et al., 2003)](image)

The matrix elements show which components share interfaces and/or design dependencies with which other components. A cross between two components in the matrix shows that they share a design interface or dependency. When the occurrence matrix is generated for matrix, each cross becomes a 1, with the interpretation that $A_{ij}$ is 1 if components $i$ and $j$ share a design interface, and is 0 if they do not. In addition, all diagonal entries $A_{ii}$ are 1 with the interpretation that a component shares a design interface with itself.

The 8 sub-systems shown in the figure have been identified by the authors of the source paper through interviews with design experts in industry practice (Sosa et al., 2003). When
REIFORM was applied onto the occurrence matrix (the six steps in Section 5.1.3), without giving it the sub-system information, it shows a “coarse” decomposition. Not only do elements from the same sub-system show high cosine measurement values with each other, but elements from different sub-systems also show high mutual cosine measurements. Identifying sub-system boundaries becomes difficult, but block matrices may still be identified in this coarse decomposition in terms of identifying sub-systems that interact strongly with each other. This leads to the identification of the modular and integrative systems, as discussed in the following sections.

5.2.1.1 Identifying intra-sub-system interactions

REIFORM was applied onto the occurrence matrix generated using the DSM in Figure 5.9 (for the occurrence matrix refer to the enclosed Matlab code; Appendix A provides the list of programs by example names). First, let us look at these “coarse” decomposition results. The diagonal blocks, (i.e. coupling strengths between components from the same sub-systems) from the $Z(2)$ matrix have been extracted for reference here. Figure 5.10 to Figure 5.17 show the coupling strength measurements between the elements of the 8 sub-systems as 8 matrices that are part of the big $54 \times 54$ matrix. These results show that REIFORM is able to identify the high intra-sub-system relationships. A standard cosine threshold of 0.5 is used to identify these sub-systems (The reason for this value will be presented in Chapter 8 where $k$ variation and cosine variation studies are presented).

<table>
<thead>
<tr>
<th>FAN</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.9988</td>
<td>0.9971</td>
<td>0.8024</td>
<td>0.8261</td>
<td>0.9948</td>
<td>0.6818</td>
<td>0.7017</td>
</tr>
<tr>
<td>2</td>
<td>0.9998</td>
<td>0.9842</td>
<td>0.7379</td>
<td>0.7648</td>
<td>0.9793</td>
<td>0.5825</td>
<td>0.6260</td>
</tr>
<tr>
<td>3</td>
<td>0.7894</td>
<td>0.8757</td>
<td>0.9983</td>
<td>0.9998</td>
<td>0.9879</td>
<td>0.9546</td>
<td>0.9775</td>
</tr>
<tr>
<td>4</td>
<td>0.6527</td>
<td>0.7832</td>
<td>0.9803</td>
<td>0.9837</td>
<td>0.7795</td>
<td>0.9777</td>
<td>0.9999</td>
</tr>
<tr>
<td>5</td>
<td>0.9835</td>
<td>0.9430</td>
<td>0.6200</td>
<td>0.6515</td>
<td>0.9342</td>
<td>0.4446</td>
<td>0.4926</td>
</tr>
<tr>
<td>6</td>
<td>0.6476</td>
<td>0.7588</td>
<td>0.9893</td>
<td>0.9825</td>
<td>0.7753</td>
<td>0.9981</td>
<td>1.0000</td>
</tr>
<tr>
<td>7</td>
<td>0.6307</td>
<td>0.7443</td>
<td>0.9859</td>
<td>0.9782</td>
<td>0.7612</td>
<td>0.9992</td>
<td>0.9999</td>
</tr>
</tbody>
</table>

Figure 5.10: Fan sub-system

<table>
<thead>
<tr>
<th>LPC</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0.9986</td>
<td>0.9999</td>
<td>0.9995</td>
<td>0.9968</td>
<td>0.8764</td>
<td>0.8172</td>
<td>0.8065</td>
</tr>
<tr>
<td>9</td>
<td>0.9986</td>
<td>0.9999</td>
<td>0.9995</td>
<td>0.9968</td>
<td>0.8764</td>
<td>0.8172</td>
<td>0.8065</td>
</tr>
<tr>
<td>10</td>
<td>0.9897</td>
<td>0.9833</td>
<td>0.9743</td>
<td>0.9932</td>
<td>0.7659</td>
<td>0.6893</td>
<td>0.6760</td>
</tr>
<tr>
<td>11</td>
<td>1.0000</td>
<td>0.9994</td>
<td>0.9969</td>
<td>0.9905</td>
<td>0.8529</td>
<td>0.7892</td>
<td>0.7779</td>
</tr>
<tr>
<td>12</td>
<td>0.8215</td>
<td>0.8432</td>
<td>0.8661</td>
<td>0.8055</td>
<td>0.9986</td>
<td>0.9982</td>
<td>0.9970</td>
</tr>
<tr>
<td>13</td>
<td>0.6573</td>
<td>0.6863</td>
<td>0.7178</td>
<td>0.6363</td>
<td>0.9556</td>
<td>0.9825</td>
<td>0.9858</td>
</tr>
<tr>
<td>14</td>
<td>0.9517</td>
<td>0.9830</td>
<td>0.9740</td>
<td>0.9430</td>
<td>0.9707</td>
<td>0.9378</td>
<td>0.9314</td>
</tr>
</tbody>
</table>

Figure 5.11: LPC sub-system
### Figure 5.12: HPC sub-system

<table>
<thead>
<tr>
<th></th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>0.9956</td>
<td>0.9961</td>
<td>0.9558</td>
<td>0.8484</td>
<td>0.9933</td>
<td>0.7935</td>
<td>0.9745</td>
</tr>
<tr>
<td>16</td>
<td>0.9851</td>
<td>0.9861</td>
<td>0.9760</td>
<td>0.8876</td>
<td>0.9810</td>
<td>0.8392</td>
<td>0.9892</td>
</tr>
<tr>
<td>17</td>
<td>0.8896</td>
<td>0.8922</td>
<td>0.9966</td>
<td>0.9843</td>
<td>0.8791</td>
<td>0.9628</td>
<td>0.9881</td>
</tr>
<tr>
<td>18</td>
<td>0.8814</td>
<td>0.8841</td>
<td>0.9950</td>
<td>0.9873</td>
<td>0.8705</td>
<td>0.9675</td>
<td>0.9852</td>
</tr>
<tr>
<td>19</td>
<td>0.9909</td>
<td>0.9915</td>
<td>0.9671</td>
<td>0.8696</td>
<td>0.9876</td>
<td>0.8181</td>
<td>0.9830</td>
</tr>
<tr>
<td>20</td>
<td>0.8114</td>
<td>0.8148</td>
<td>0.9732</td>
<td>0.9966</td>
<td>0.7980</td>
<td>0.9024</td>
<td>0.9540</td>
</tr>
<tr>
<td>21</td>
<td>0.9293</td>
<td>0.9315</td>
<td>0.9999</td>
<td>0.9629</td>
<td>0.9237</td>
<td>0.9325</td>
<td>0.9983</td>
</tr>
</tbody>
</table>

### Figure 5.13: CC sub-system

<table>
<thead>
<tr>
<th></th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>0.9686</td>
<td>0.9947</td>
<td>0.9437</td>
<td>0.9960</td>
<td>0.9975</td>
</tr>
<tr>
<td>23</td>
<td>0.9858</td>
<td>0.9953</td>
<td>0.9416</td>
<td>0.9976</td>
<td>0.9979</td>
</tr>
<tr>
<td>24</td>
<td>0.9648</td>
<td>0.9047</td>
<td>1.0000</td>
<td>0.9626</td>
<td>0.8180</td>
</tr>
<tr>
<td>25</td>
<td>0.9735</td>
<td>0.9994</td>
<td>0.9190</td>
<td>0.9914</td>
<td>1.0000</td>
</tr>
<tr>
<td>26</td>
<td>0.9928</td>
<td>0.9894</td>
<td>0.9569</td>
<td>0.9998</td>
<td>0.9936</td>
</tr>
</tbody>
</table>

### Figure 5.14: HPT sub-system

<table>
<thead>
<tr>
<th></th>
<th>27</th>
<th>28</th>
<th>29</th>
<th>30</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>0.9440</td>
<td>0.9684</td>
<td>0.9749</td>
<td>0.9925</td>
<td>0.9958</td>
</tr>
<tr>
<td>28</td>
<td>0.9959</td>
<td>1.0000</td>
<td>0.9980</td>
<td>0.9924</td>
<td>0.9436</td>
</tr>
<tr>
<td>29</td>
<td>0.8651</td>
<td>0.9212</td>
<td>0.9315</td>
<td>0.9837</td>
<td>0.9965</td>
</tr>
<tr>
<td>30</td>
<td>0.9712</td>
<td>0.9879</td>
<td>0.9917</td>
<td>0.9996</td>
<td>0.9825</td>
</tr>
<tr>
<td>31</td>
<td>0.9623</td>
<td>0.9818</td>
<td>0.9866</td>
<td>0.9981</td>
<td>0.9883</td>
</tr>
</tbody>
</table>

### Figure 5.15: LPT sub-system

<table>
<thead>
<tr>
<th></th>
<th>32</th>
<th>33</th>
<th>34</th>
<th>35</th>
<th>36</th>
<th>37</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>0.9284</td>
<td>0.9729</td>
<td>0.9918</td>
<td>0.7779</td>
<td>0.7798</td>
<td>0.7760</td>
</tr>
<tr>
<td>33</td>
<td>0.9999</td>
<td>0.9908</td>
<td>0.9713</td>
<td>0.9521</td>
<td>0.9530</td>
<td>0.9512</td>
</tr>
<tr>
<td>34</td>
<td>0.9947</td>
<td>0.9990</td>
<td>0.9888</td>
<td>0.9205</td>
<td>0.9217</td>
<td>0.9193</td>
</tr>
<tr>
<td>35</td>
<td>0.9232</td>
<td>0.8567</td>
<td>0.7982</td>
<td>0.9954</td>
<td>0.9951</td>
<td>0.9957</td>
</tr>
<tr>
<td>36</td>
<td>0.9701</td>
<td>0.9238</td>
<td>0.9706</td>
<td>0.9985</td>
<td>0.9987</td>
<td>0.9984</td>
</tr>
<tr>
<td>37</td>
<td>0.9666</td>
<td>0.9185</td>
<td>0.8721</td>
<td>0.9992</td>
<td>0.9993</td>
<td>0.9991</td>
</tr>
</tbody>
</table>

### Figure 5.16: Mechanicals sub-system

<table>
<thead>
<tr>
<th>Mech</th>
<th>38</th>
<th>39</th>
<th>40</th>
<th>41</th>
<th>42</th>
<th>43</th>
<th>44</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>0.9563</td>
<td>0.9921</td>
<td>0.9948</td>
<td>0.9947</td>
<td>0.9946</td>
<td>0.9981</td>
<td>0.9995</td>
</tr>
<tr>
<td>39</td>
<td>0.9338</td>
<td>0.9984</td>
<td>0.9854</td>
<td>0.9851</td>
<td>0.9850</td>
<td>0.9915</td>
<td>0.9960</td>
</tr>
<tr>
<td>40</td>
<td>0.9803</td>
<td>0.9750</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>0.9993</td>
<td>0.9977</td>
</tr>
<tr>
<td>41</td>
<td>0.9881</td>
<td>0.9642</td>
<td>0.9992</td>
<td>0.9992</td>
<td>0.9993</td>
<td>0.9967</td>
<td>0.9938</td>
</tr>
<tr>
<td>42</td>
<td>0.9032</td>
<td>0.9998</td>
<td>0.9692</td>
<td>0.9688</td>
<td>0.9686</td>
<td>0.9783</td>
<td>0.9843</td>
</tr>
<tr>
<td>43</td>
<td>0.9735</td>
<td>0.9817</td>
<td>0.9994</td>
<td>0.9993</td>
<td>0.9993</td>
<td>1.0000</td>
<td>0.9994</td>
</tr>
<tr>
<td>44</td>
<td>0.9543</td>
<td>0.9929</td>
<td>0.9941</td>
<td>0.9939</td>
<td>0.9938</td>
<td>0.9977</td>
<td>0.9993</td>
</tr>
</tbody>
</table>
5.2.1.2 Identifying inter-sub-system couplings

The matrix reordering produced by using the cosine values as discussed above produces a “coarse” decomposition because REIFORM shows high interaction relationships between components from the same as well as different sub-systems that share strong interactions. For example, cosine measurements between the Fan and LPC sub-systems come out to be high (Figure 5.18 and Figure 5.19) because many components of the Fan sub-system interact with many components of the LPC sub-system (Figure 5.9). This is the basis for identifying the modular and integrative sub-systems.

For the aero-engine problem, components of one sub-system share design interfaces with components from other sub-systems. In such a case, modularity is defined on the basis of the observation that sub-systems that share concentrated interactions with only a few other sub-systems (for instance, on account of spatial integrity) will be defined as modular. On the other hand, sub-systems that show distributed interactions with all other subsystems will be defined as integrative. For example, the Low Pressure Compressor (LPC) subsystem is a modular sub-system because its components share design interfaces with components from the Fan and the High Pressure Compressor (HPC) sub-systems but not with the others (Sosa et al., 2003). On the other hand, the externals and controls sub-system is an integrative one because its components shares design interactions with components from all the other subsystems (Sosa et al., 2003).

Following up this example in terms of REIFORM, in the “coarse” decomposition, the LPC system components should show high coupling strengths, i.e. high cosine similarity with the Fan and the HPC system components but low cosine similarity with the other systems. Figure 5.18 to Figure 5.21 show that this is indeed the case – the LPC-Fan, Fan-LPC, LPC-HPC and HPC-LPC matrices show high cosine measurements. Note that the measurements for a system pair lead to two different matrices because the original matrix contains asymmetric design relations. Thus, for example, the Fan-LPC (rows 1-7 and columns 8-14 in

<table>
<thead>
<tr>
<th>E/C</th>
<th>45</th>
<th>46</th>
<th>47</th>
<th>48</th>
<th>49</th>
<th>50</th>
<th>51</th>
<th>52</th>
<th>53</th>
<th>54</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>0.9911</td>
<td>0.8326</td>
<td>0.9981</td>
<td>0.9997</td>
<td>0.9997</td>
<td>0.9321</td>
<td>0.9679</td>
<td>0.9349</td>
<td>1.0000</td>
<td>0.9550</td>
</tr>
<tr>
<td>46</td>
<td>0.9294</td>
<td>0.9419</td>
<td>0.9536</td>
<td>0.9644</td>
<td>0.9646</td>
<td>0.8168</td>
<td>0.8785</td>
<td>0.8213</td>
<td>0.9662</td>
<td>0.8548</td>
</tr>
<tr>
<td>47</td>
<td>0.9994</td>
<td>0.7737</td>
<td>0.9933</td>
<td>0.9971</td>
<td>0.9971</td>
<td>0.9634</td>
<td>0.9681</td>
<td>0.9655</td>
<td>0.9959</td>
<td>0.9797</td>
</tr>
<tr>
<td>48</td>
<td>1.0000</td>
<td>0.7525</td>
<td>0.9975</td>
<td>0.9941</td>
<td>0.9940</td>
<td>0.9717</td>
<td>0.9926</td>
<td>0.9735</td>
<td>0.9924</td>
<td>0.9857</td>
</tr>
<tr>
<td>49</td>
<td>0.9733</td>
<td>0.8828</td>
<td>0.9873</td>
<td>0.9926</td>
<td>0.9927</td>
<td>0.8922</td>
<td>0.9388</td>
<td>0.8957</td>
<td>0.9944</td>
<td>0.9214</td>
</tr>
<tr>
<td>50</td>
<td>0.9617</td>
<td>0.5417</td>
<td>0.9395</td>
<td>0.9257</td>
<td>0.9254</td>
<td>0.9992</td>
<td>0.9876</td>
<td>0.9888</td>
<td>0.9199</td>
<td>0.9940</td>
</tr>
<tr>
<td>51</td>
<td>0.9929</td>
<td>0.8244</td>
<td>0.9969</td>
<td>1.0000</td>
<td>1.0000</td>
<td>0.9373</td>
<td>0.9715</td>
<td>0.9400</td>
<td>1.0000</td>
<td>0.9592</td>
</tr>
<tr>
<td>52</td>
<td>0.9986</td>
<td>0.7149</td>
<td>0.9921</td>
<td>0.9866</td>
<td>0.9865</td>
<td>0.9632</td>
<td>0.9976</td>
<td>0.9847</td>
<td>0.9840</td>
<td>0.9935</td>
</tr>
<tr>
<td>53</td>
<td>0.9714</td>
<td>0.8866</td>
<td>0.9860</td>
<td>0.9916</td>
<td>0.9917</td>
<td>0.8884</td>
<td>0.9359</td>
<td>0.8920</td>
<td>0.9934</td>
<td>0.9181</td>
</tr>
<tr>
<td>54</td>
<td>0.9940</td>
<td>0.8193</td>
<td>0.9963</td>
<td>1.0000</td>
<td>1.0000</td>
<td>0.9404</td>
<td>0.9736</td>
<td>0.9431</td>
<td>0.9999</td>
<td>0.9617</td>
</tr>
</tbody>
</table>

Figure 5.17: Externals and controls sub-system
relationships could be different from the LPC-Fan (rows 8-14 and columns 1-7 in $Z(2)$) relationships.

![Figure 5.18: Fan-LPC cosine measurements](image)

<table>
<thead>
<tr>
<th>LPC-LPC</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.9292</td>
<td>0.8633</td>
<td>0.9967</td>
<td>0.9962</td>
<td>0.8979</td>
<td>0.9585</td>
<td>0.9726</td>
</tr>
<tr>
<td>2</td>
<td>0.9292</td>
<td>0.8633</td>
<td>0.9967</td>
<td>0.9962</td>
<td>0.8979</td>
<td>0.9585</td>
<td>0.9726</td>
</tr>
<tr>
<td>3</td>
<td>0.6715</td>
<td>0.7792</td>
<td>0.9934</td>
<td>0.9879</td>
<td>0.7950</td>
<td>0.9566</td>
<td>0.9933</td>
</tr>
<tr>
<td>4</td>
<td>0.7741</td>
<td>0.8634</td>
<td>0.9954</td>
<td>1.0000</td>
<td>0.8762</td>
<td>0.9708</td>
<td>0.9824</td>
</tr>
<tr>
<td>5</td>
<td>0.9065</td>
<td>0.9973</td>
<td>0.9045</td>
<td>0.8281</td>
<td>0.9951</td>
<td>0.8645</td>
<td>0.7042</td>
</tr>
<tr>
<td>6</td>
<td>0.9868</td>
<td>0.9963</td>
<td>0.9350</td>
<td>0.8660</td>
<td>0.9409</td>
<td>0.4618</td>
<td>0.5995</td>
</tr>
<tr>
<td>7</td>
<td>0.9291</td>
<td>0.9755</td>
<td>0.9424</td>
<td>0.9553</td>
<td>0.9809</td>
<td>0.8530</td>
<td>0.8802</td>
</tr>
</tbody>
</table>

![Figure 5.19: LPC-Fan cosine measurements](image)

Figure 5.20 to Figure 5.23 show, as examples, that the LPC-HPT and HPT-LPC matrices show low mutual cosine similarity, as it does not share design interfaces with these systems. From the detailed results, the LPC sub-system shows low cosine similarity with HPT, LPT and CC systems.
5.2.1.3 Identifying the modular and integrative sub-systems

At \( k=2 \), the Fan, LPC, and HPC systems show high cosine measurements with each other, and the HPT, LPT and CC systems show high cosine measurements with each other. That is, two block matrices are identified in the “coarse” decomposition using the matrix partitioning approach, identifying the Fan, LPC, HPC, CC, HPT and LPT as modular systems. Mechanicals and Externals systems show generally high cosines with all the systems identifying them as the integrative sub-systems. To confirm these results, a K-means analysis was done on the vector representation of components in 2D space. Figure 5.24 shows the results for components 1 – 37 (Fan, LPC, HPC, CC, HPT, LPT). The number of clusters was chosen to be 2 in order to check whether K-means algorithm also returns a similar clustering as the matrix partitioning analysis. In addition, the visual interpretation helped. Figure 5.24 shows the same results – Components 1 – 21 (Fan, LPC, HPC) were put into one cluster and components 22 – 37 (CC, HPT, LPT) were put into another cluster. The Mechanical and Externals/Controls components are not shown for clarity of presentation in the graph here because they tend to be distributed across the clusters.
In this way, an analysis of the $54 \times 54$ cosine measurement matrix can reveal the modular versus the integrative subsystems. However, for a complex problem such as this, although the $k$ reduced matrix shows the patterns of modularity and integration, it will be better to develop another concise metric for interpreting and viewing the result. Here, the matrix norm (the largest singular value of a matrix) is chosen as that measure and the same decomposition definition is applied onto this condensed matrix. If each of the “blocks” that signifies subsystem interactions between components is chosen as one matrix (consider these tables as examples of “blocks”), then the $54 \times 54$ matrix will have 64 such blocks (8 sub-systems interacting with 8 sub-systems). Then, if each of these 64 matrices is replaced with its matrix norm, the $54 \times 54$ matrix collapses into an $8 \times 8$ matrix, each row or column representing a sub-system. Figure 5.25 shows the results of doing this for $k = 2$ reduced matrix. Now if 3.0 is chosen as the threshold value (the threshold is the norm value, and not the cosine threshold), this confirms the earlier analysis – the Fan, LPC, and HPC systems form one block matrix and the CC, HPT and LPT systems form another. The Mechanicals and Externals/Controls systems show integrative relationships with all sub-systems. The last row (TOTAL) shows the number of other sub-systems a particular sub-system has high interactions with beyond the threshold. A low number indicates a modular system; the highest numbers indicate integrative systems.
As the $k$ values are increased, HPC is identified as somewhat less modular than the other identified modular systems, but also not as integrative as the mechanicals and externals and controls system. This matches with the analysis in the original paper. There is a discussion by Rowles questioning whether it is a modular or an integrative system, and it is finally identified as a modular one.

The threshold values and $k$ values for performing dimensionality reduction seem arbitrary here. These results present the “good” results from trials involving heuristically guided choices. Chapter 8 presents heuristics for choosing good parametric values for cosine thresholds for the cosine measurement analysis, number of clusters for the K-means analysis, and the number of dimensions $k$ to be retained in the dimensionality reduction step. These are the three of the ways in which a designer may experiment with different decisions using the same problem representation. Figure 5.26 shows the results for $k=54$, i.e. the other end of approximation. These values will provide the explicit information exactly as exists in the occurrence matrix. Although the decision on modular versus integrative systems does not change, the patterns are different. The Fan, LPC, CC, HPT and LPT systems are the most modular, HPC shows less modularity than these systems but more than the Mechanicals and Externals/Controls system, and these latter ones are integrative.

<table>
<thead>
<tr>
<th>$k=2$</th>
<th>Fan</th>
<th>LPC</th>
<th>HPC</th>
<th>CC</th>
<th>HPT</th>
<th>LPT</th>
<th>Mech</th>
<th>Ext/Con</th>
<th>Total (Rows)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAN</td>
<td>5.97</td>
<td>6.05</td>
<td>6.06</td>
<td>2.49</td>
<td>2.58</td>
<td>2.73</td>
<td>3.32</td>
<td>4.56</td>
<td>5</td>
</tr>
<tr>
<td>LPC</td>
<td>6.22</td>
<td>6.31</td>
<td>6.32</td>
<td>2.78</td>
<td>2.07</td>
<td>2.17</td>
<td>2.91</td>
<td>4.38</td>
<td>4</td>
</tr>
<tr>
<td>HPC</td>
<td>6.36</td>
<td>6.53</td>
<td>6.57</td>
<td>1.91</td>
<td>1.11</td>
<td>1.48</td>
<td>3.19</td>
<td>4.98</td>
<td>5</td>
</tr>
<tr>
<td>CC</td>
<td>2.05</td>
<td>2.02</td>
<td>2.11</td>
<td>4.89</td>
<td>4.72</td>
<td>5.16</td>
<td>5.74</td>
<td>6.42</td>
<td>5</td>
</tr>
<tr>
<td>HPT</td>
<td>2.07</td>
<td>1.77</td>
<td>1.74</td>
<td>4.85</td>
<td>4.87</td>
<td>5.28</td>
<td>5.53</td>
<td>6.37</td>
<td>5</td>
</tr>
<tr>
<td>LPT</td>
<td>2.37</td>
<td>1.93</td>
<td>1.81</td>
<td>5.13</td>
<td>5.23</td>
<td>5.65</td>
<td>5.80</td>
<td>6.65</td>
<td>5</td>
</tr>
<tr>
<td>Mech</td>
<td>3.06</td>
<td>3.32</td>
<td>3.51</td>
<td>5.63</td>
<td>5.09</td>
<td>5.65</td>
<td>6.91</td>
<td>8.03</td>
<td>8</td>
</tr>
<tr>
<td>Ext/Con</td>
<td>4.49</td>
<td>4.93</td>
<td>5.16</td>
<td>6.28</td>
<td>5.39</td>
<td>6.08</td>
<td>7.96</td>
<td>9.52</td>
<td>8</td>
</tr>
<tr>
<td>Total (Cols)</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>10</td>
<td>9</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>15</td>
<td>16</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 5.25: Modularity and integration assessment results, $k=2$**

<table>
<thead>
<tr>
<th>$k=54$</th>
<th>Fan</th>
<th>LPC</th>
<th>HPC</th>
<th>CC</th>
<th>HPT</th>
<th>LPT</th>
<th>Mech</th>
<th>Ext/Con</th>
<th>Total (Rows)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAN</td>
<td>3.73</td>
<td>2.41</td>
<td>1.09</td>
<td>0.45</td>
<td>0.37</td>
<td>0.71</td>
<td>1.35</td>
<td>1.95</td>
<td>5</td>
</tr>
<tr>
<td>LPC</td>
<td>2.82</td>
<td>3.68</td>
<td>2.97</td>
<td>0.65</td>
<td>0.36</td>
<td>0.49</td>
<td>0.93</td>
<td>2.61</td>
<td>4</td>
</tr>
<tr>
<td>HPC</td>
<td>2.02</td>
<td>2.85</td>
<td>4.45</td>
<td>1.09</td>
<td>0.68</td>
<td>0.50</td>
<td>1.42</td>
<td>2.31</td>
<td>5</td>
</tr>
<tr>
<td>CC</td>
<td>0.53</td>
<td>0.76</td>
<td>1.04</td>
<td>2.76</td>
<td>1.63</td>
<td>0.71</td>
<td>1.81</td>
<td>2.96</td>
<td>5</td>
</tr>
<tr>
<td>HPT</td>
<td>0.38</td>
<td>0.49</td>
<td>1.11</td>
<td>1.92</td>
<td>3.50</td>
<td>1.95</td>
<td>0.90</td>
<td>2.09</td>
<td>5</td>
</tr>
<tr>
<td>LPT</td>
<td>0.59</td>
<td>0.43</td>
<td>0.95</td>
<td>0.57</td>
<td>1.78</td>
<td>3.76</td>
<td>1.23</td>
<td>1.78</td>
<td>4</td>
</tr>
<tr>
<td>Mech</td>
<td>1.29</td>
<td>1.49</td>
<td>1.31</td>
<td>2.01</td>
<td>0.89</td>
<td>1.45</td>
<td>3.80</td>
<td>3.57</td>
<td>7</td>
</tr>
<tr>
<td>Ext/Con</td>
<td>2.23</td>
<td>3.07</td>
<td>2.35</td>
<td>2.78</td>
<td>1.72</td>
<td>2.18</td>
<td>3.59</td>
<td>6.37</td>
<td>8</td>
</tr>
<tr>
<td>Total (Cols)</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>10</td>
<td>9</td>
<td>12</td>
<td>9</td>
<td>8</td>
<td>13</td>
<td>14</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 5.26: Modularity and integration assessment results, $k=54$**
5.3 Evaluation of REIFORM: Aircraft Concept Sizing (ACS) problem

In this final section REIFORM is applied to an Aircraft Concept Sizing (ACS) problem as a validation experiment. The ACS problem was chosen because the analysis could run in the opposite direction – given a problem for which the cases or sub-problems are already known, and given the condition that it is a problem where a significant amount of coupling exists between the variables, will REIFORM be able to identify the “correct” cases or sub-problems? This section presents the result of this evaluation.

The source for the ACS problem is Gu et al. (2002). Gu formulates it as a decision based collaborative optimization problem, bringing in collaborative optimization (CO), decision based design (DBD) and multidisciplinary optimization (MDO) in a single framework. Figure 5.27, Figure 5.28 and Figure 5.29 show the formulation of the problem, with 8 variables ($x_1$ – $x_8$), 10 system states ($x_9$ – $x_{18}$) and 5 sub-problems (SP1 – SP5) that represent 5 domains (sub problems) (aerodynamics, weight, performance, cost and business). System states identify couplings between the design sub problems, as outputs from one sub-domain can be inputs to other domains. The problem has been solved numerically in their paper using a Sequential Quadratic Programming method. There is no attempt here to solve this problem numerically; therefore the symbolic model is not being assessed for its “formulation quality”. The focus for REIFORM is primarily symbolic modeling and reformulation, for which this experiment validates whether REIFORM returns the sub-problem decomposition SP1 to SP5 as described by Gu, when all the interaction information is merged and used as input into REIFORM. Thus, the evaluation tests whether the method will return the cases as identified in the original problem without “telling” it that these cases exist, but giving it only the interaction/coupling information.

The objective of the main problem is to maximize the Net Revenue (or minimize the negative of the Net Revenue). The main problem contains compatibility constraints that become objectives for the sub-problems.

<table>
<thead>
<tr>
<th>Design Variables</th>
<th>Description</th>
<th>Design states</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>Aspect ratio of wing</td>
<td>$x_9$</td>
<td>Total aircraft wetted area (ft$^2$)</td>
</tr>
<tr>
<td>$x_2$</td>
<td>Wing area (ft$^2$)</td>
<td>$x_{10}$</td>
<td>Max lift to drag ratio</td>
</tr>
<tr>
<td>$x_3$</td>
<td>Fuselage length (ft)</td>
<td>$x_{11}$</td>
<td>Empty weight (lbs)</td>
</tr>
<tr>
<td>$x_4$</td>
<td>Fuselage diameter (ft)</td>
<td>$x_{12}$</td>
<td>Gross take off weight (lbs)</td>
</tr>
<tr>
<td>$x_5$</td>
<td>Density of air cruise altitude (slugs/ft$^2$)</td>
<td>$x_{13}$</td>
<td>Aircraft range (miles)</td>
</tr>
<tr>
<td>$x_6$</td>
<td>Cruise speed (ft/sec)</td>
<td>$x_{14}$</td>
<td>Stall speed (ft/sec)</td>
</tr>
<tr>
<td>$x_7$</td>
<td>Fuel weight (lbs)</td>
<td>$x_{15}$</td>
<td>Fuselage volume</td>
</tr>
<tr>
<td>$x_8$</td>
<td>Price</td>
<td>$x_{16}$</td>
<td>Demand</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$x_{17}$</td>
<td>Total cost</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$x_{18}$</td>
<td>Net Revenue</td>
</tr>
</tbody>
</table>

Figure 5.27: Design variables and states for the ACS problem
REIFORM was applied to this problem, varying the number of dimensions from 2 to 4 \( (k = 2, 3, 4) \) in order to study the variations in the number of correct cases returned and the number of missed cases not returned by the method. Correct cases are defined as those variables or design states that are part of the original sub-problem definition and should be returned by the method. Missed cases are those that are correct but not returned as an answer. As such, correct cases measure the success, and missed cases measure failure of the method to classify a variable or state into the “correct” sub-problem. The method should produce groupings of shared and independent variables, i.e., which variables and states are shared or occur independently in which sub-problems. Because this is a medium-large scale complex problem, another related aim in this experiment was to study at what reduced dimensionality the method successfully returns semantic groupings as expressed in the problem statement. Note that the problem is characterized by each of the sub-problems sharing many design variables and system state variables. This implies that there will be many explicit as well as implicit relationships between the sub-problems because of these couplings.

The results show that REIFORM captures at \( k = 2 \) or 3 almost all the correct groupings, with very few missed cases. Performance slightly improves over this initial case at \( k = 4 \), showing that the reduced dimensionality representations capture the relationships between variables, states and sub-problems correctly. The method, especially at \( k = 2 \), also returns some extra cases, i.e. variables or system states that are not part of the original sub-problem, but show a high interaction relationship with the given sub-problem. We cannot classify these extra cases as “incorrect”, as from previous experiments we know that the method returns cases on the basis of relationships that both occur explicitly or latently in the problem syntax.
Therefore, these “extra” cases show some implied relationships that are not explicit in the formulation. These results suggest that an alternate formulation for the design problem is possible where the implied associative relationships could be made explicit. This alternate formulation could be run through a numerical optimizer to determine whether it is less costly to optimize. Because the implied relationships capture the strong “coupling” variables and states, making an implied one explicit could reduce the computational cost due to coordinating sub-problem solutions.

In REIFORM, the $k$-reduced approximation is a linear least squares approximation of the original occurrence matrix, but unlike traditional “error” reduction approaches, a larger error in this case can be useful. As $k$ decreases ($k = n, n-1, n-2,..., 1$) the degree of error will increase, but the size of this error may result in latent relationships that were not perceivable when the error is small. It is not a priori obvious if there is an optimal error. For example, calculating the estimation error for cases $k = 2, 3, 4$ and $5$, by using an infinity norm (largest row sum for the matrix), between the original matrix and the $k$-reduced approximations shows the following – for the original matrix, the norm is 4, for the 2-reduced matrix the norm is 3.8496, and for the 3-reduced matrix the norm is 3.9589, for the 4-reduced matrix the norm is 3.9404, for the 5-reduced (or original matrix) the norm is 4.0000 again. As expected, the error reduces from 0.1504 to 0 when $k$ increases from 2 to 5. However, as observed above, the results obtained in terms of semantic groupings for the $k = 2$ or 3 case were more, or at least as, relevant in terms of semantic groupings obtained for variables and constraints as for the $k = 4$ or 5 cases. Figure 5.30 shows the details of the correct, extra and missed variables and states returned by the method for $k = 2, 3, 4$ with a cosine threshold of 0.7, and the queries set to the 5 sub-problems.
Two metrics were used to measure the performance of the method – a precision metric and a recall metric. Precision is defined as the ratio of correct cases to total cases found. Recall is defined as the ratio of correct cases to the total number of correct cases that should have been found. Figure 5.31 shows the improvement in performance for $k = 2$ (i.e., Precision$_{2D}$ and Recall$_{2D}$) and $k = 3$ and then a similar level of performance for $k = 4$. At $k = 5$, the matrix becomes the original matrix; Precision and Recall become 1. But, even at $k = 2$ or $k = 3$, the answers returned by the method capture the patterns of variable-state clusters in the problem. This is a significant advantage for complex problems.
5.4 Summary

This chapter presented the application of REIFORM for decomposition and modularity analysis tasks. The 4 steps of REIFORM were used and extended with a matrix partitioning and an unsupervised clustering approach to handle problem decomposition and modularity analysis. FDT and DSM representations were used as examples for demonstration. It was shown that the basis of using REIFORM for problem decomposition and modularity analysis lies in the way it converts discrete interaction strength data into a continuous distance based representation, and induces patterns of implied interactions between variables, functions and system components across various dimensions. A validation experiment was performed where the “correct” answers were pre-defined, and REIFORM’s performance was tested using precision and recall metrics.
Chapter 6

Topology design: Space Layout Planning

The substratum is that which continues to exist and maintain its characteristic quality, whether manifest, latent, or subdued.

Yoga Sutras, Patanjali

Do I contradict myself? Very well then, I do. I am large. I contain multitudes.

Walt Whitman, Song of Myself

This chapter presents the application of REIFORM for topology design as a constraint satisfaction type of design reformulation task. Problems such as topology modeling have an inherent combinatorial, discrete, multi-modal nature and may have parts that are difficult to model analytically. Therefore, it may be beneficial to engage in model reformulation and refinement before and while a final detailed analytical model is being developed. REIFORM is applied to architectural design examples to demonstrate topology modeling.

System decomposition and modularity analysis, the tasks presented in Chapter 5, are different types of reformulation tasks as compared to topology modeling. However, this chapter demonstrates that, despite the objective of the reformulation tasks being different in these cases, REIFORM uses the same 4 steps as described in Chapter 4 to perform both types of reformulation tasks. The choice of tasks in this chapter and Chapter 5 is not exhaustive in any sense. They were chosen to illustrate the various ways in which the same method structure could be used to perform different kinds of reformulation tasks. The tasks may be different, but they share one common structural similarity, and this is the basis for the application of REIFORM – the local and global “meaning” contained in the associative relationships between symbols is the basis for reformulation.
6.1 Topology modeling and optimization

Topology optimization is a process in which a feasible or best configuration has to be found for a set of components, given some constraints on connectivity or adjacency. For example, if we see space layout planning as a topology modeling and optimization problem, then a set of constraints on connectivity or adjacency is specified between a set of spaces. The problem is to work out the feasible or best configurations that do not violate these constraints.

In topology modeling in some design domains, one characteristic to note is that an explicit set of constraints between components may introduce implicit sets of constraints between components. For example, consider the following simple set of space related constraints that an architect may often employ in house design problems: (a) the dining room should be directly connected to the kitchen, and may be connected to the living room; (b) the living room should definitely be connected to the bedrooms; (c) the bedrooms should not be connected to the kitchen, and should be definitely connected to the bathrooms. This explicit set of constraints generates an implicit constraint: the bedrooms and bathrooms should not be connected to the dining room. This is not explicitly stated, but follows from the three explicitly stated constraints. A method that can infer such implicit constraints from a set of explicit constraints will be able to generate feasible topology solutions.

This is the kind of implicit mapping that REIFORM should be able to infer. It has been demonstrated in Chapter 4 that the SVD and dimensionality reduction steps induce a set of implicit relationships using the set of explicit relationships that are specified in the occurrence matrix using all possible linear combinations of the original matrix entries to generate linearly independent vectors to represent the components. The values in the approximations approach the limit set by the explicit occurrence matrix entries. The $k$ largest singular values capture the strongest association patterns in the matrix. Thus, these cause the strongly connected parts of a system to come close together in the reduced dimension space, and cause the weakly connected parts to be pushed apart. In topology, the relational knowledge that the occurrence matrix captures is actually constraint knowledge that exists between parts of a system. Continuing with the example above, while REIFORM will infer that the kitchen-living-dining should be a tight cluster, and the bedroom-bathroom should be a tight cluster, with the living providing the connection between these two local clusters. Thus, logically, the set of implicit approximated relationships generated can never violate the set of explicit relationships because the explicit ones lie at the basis of inferring the implicit ones. The implicit ones, though, can reinforce some of the stronger association patterns in the explicit matrix. Following the same logic, it can also be argued that since the $k$-reduced approximations are all approximations to the original set of relationships, different sets of $k$ values will show different implicit relationship structures. Some of these will lead to well-
formed solutions. Thus, this chapter demonstrates that REIFORM can be used as a problem reformulation technique that performs constraint satisfaction between mutually related and constrained sets of components to generate feasible topologies or layouts.

Different from the matrix partitioning or clustering approach, in this type of task, Step IV of REIFORM (cosine similarity measurement step) is used to infer the feasible topology. The following steps are used for generating feasible topologies:

1. In Step I, a designer creates an occurrence matrix. The matrix specifies the various constraints between the components in terms of a binary mapping or numerical values. A binary mapping implies that a connection does / does not exist between two components. A numerical (for example integer) mapping implies the strength of connectivity relationships that exists between the components. Since the SVD can be computed on both, both forms are permissible.

2. In Steps II and III, the SVD and dimensionality reduction computation on the occurrence matrix identifies the implicit set of associative relationships as generated by the explicit set of relationships in the original occurrence matrix $A$. For this particular task, the associative relations are actually constraints. The $k$ values are varied and implicit relationships are observed for a range of $k$ values. As described in Chapter 4, the singular values capture the most important associative patterns in the occurrence matrix in decreasing order of magnitude. Therefore, if a parametric study is done by varying the $k$ approximations and observing the resulting implicit relationships, a certain range of $k$ values can be identified that generates well-formed solutions, in this case, feasible topologies. As the $k$ values are increased, the implicit relationships approach the limit set by the explicit ones because the approximations start approaching the original occurrence matrix. At a certain $k$ value, the information returned is exactly the same as the occurrence matrix, i.e. the original explicit relationships.

3. In Step IV, for a particular $k$-reduced approximation, a cosine threshold value is chosen. Any cosine measurement between two components that is beyond the cosine threshold is interpreted to be a connectivity link between these two components. In this manner, by varying the cosine threshold, different topologies can be observed.

The next section demonstrates this method for feasible topology generation for two space layout examples.

6.2 Space layout planning: topology modeling

This section demonstrates the application of REIFORM for topology modeling. An illustrative house layout problem adapted from Michalek et al. (2002) is used to demonstrate how REIFORM can be used for modeling topology. Then, the same problem is expanded to
define a large floor plan layout problem comprised by three housing units and common circulation space, also adapted from Michalek et al. (2002), to demonstrate the approach on a larger problem.

6.2.1 House layout problem: topology modeling using REIFORM

In this section, an illustrative house layout problem is described, adapted from Michalek et al. (2002). Michalek et al. approach the problem from a combined geometry and topology optimization perspective. This section focuses solely on topology. Though a combined geometry and topology optimization approach is powerful, architects frequently generate concept or “bubble” diagrams prior to taking decisions on geometry. For many kinds of architectural design problems, relationships of desirable or undesirable connectivity between spaces are the basis for generating the final geometry. Therefore, the main focus is to show how REIFORM can be used as a topology modeling tool that allows a designer to explore quick redefinitions of a problem to explore topologies. Topology modeling uses REIFORM in a different way from its use in problem decomposition type problems. The method structure of REIFORM does not change, but the way it is interpreted in its use as a problem reformulation method changes. As described in the previous section, it is used as a problem reformulation technique that performs explicit and implicit constraint satisfaction between mutually related and constrained sets of spaces and activities to generate feasible topologies or layouts.

The problem description used in this section is changed slightly from the form as presented by Michalek et al. The authors of the source paper formulate the problem in analytical form, while REIFORM requires a non-analytic description. For this reason, some of the constraints defined by Michalek et al. are not required for REIFORM, or stated differently. In addition, the architectural “domain” constraints have been changed slightly in form due to functional and aesthetic considerations. For example, desirable or undesirable connections can actually be graded with a numerical value, rather than simply stating them as a connectivity (1) or no connectivity (0) relationship.

The problem is described as follows. A one bedroom apartment is to be designed. It contains 5 types of spaces – living, dining, kitchen, bedroom, and bathroom. There are desirable and undesirable connectivity and adjacency constraints between these spaces. A topology is to be generated for connections between spaces that respect the desirability or undesirability criteria outlined as requirements. The main characteristic to note in this problem is that imposing an explicit criterion between spaces introduces some implicit constraints between them. It is due to this specific problem characteristic that REIFORM is suitable to apply – it can extract implicit relationships from explicit constraints, Further, note that too many explicit constraints in these types of problems can often lead to over-
constrained, ill-formed problems or the nature of the explicit relationship specified may conflict with some other implicit constraint generated out other explicit criteria. REIFORM will be used here as a modeling and reformulation tool to identify feasible topologies.

The matrix form developed for REIFORM is a DSM form, in which each row and column represents a particular space, and the matrix entries capture the connectivity / adjacency relationships between these spaces. Consider the following set of constraints:

1. Each space in the apartment has the highest connectivity to its own self.
   Diagonal scores in the matrix = 4
2. In each apartment, there must be a path from the kitchen to the dining room that may pass through the living room.
   Kitchen-dining score = 3
   Kitchen-living score = 1
3. In each apartment, there must be a path from the bathroom to the living room that may not pass through the kitchen.
   Living – Bath score = 3
   Kitchen – Bath score = -1
4. In each apartment, there must be a path from the bedroom to the living room that may pass through the dining room.
   Bedroom – Living score = 3
   Bedroom – Dining score = 1

Figure 6.1 shows the matrix form for capturing these relationships.

<table>
<thead>
<tr>
<th></th>
<th>Living</th>
<th>Dining</th>
<th>Kitchen</th>
<th>Bedroom</th>
<th>Bathroom</th>
</tr>
</thead>
<tbody>
<tr>
<td>Living</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Dining</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Kitchen</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>Bedroom</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Bathroom</td>
<td>3</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>

Figure 6.1: House Layout Problem Occurrence Matrix

Note that, in the modeling stage, many of these relationships arise from functional and behavioral requirements (for e.g. dining room being adjacent to the kitchen), but many can arise from subjective or aesthetic considerations (for e.g. the client “culturally” believes that bathrooms should necessarily be away from all food preparation and eating areas of the house). REIFORM provides a common representation form for encoding both functional/behavioral and/or subjective/aesthetic constraints – the occurrence matrix form can merge these and encode these constraints in a single matrix. As previously noted, REIFORM also provides a quick and efficient way of allowing interactive modeling – the designer can change the matrix entries to observe feasible topologies.
Steps II, III and IV of REIFORM are applied onto the matrix shown in Figure 6.1. Figure 6.2 shows the cosine measurements existing between any two elements, with $k=2$.

<table>
<thead>
<tr>
<th></th>
<th>Living</th>
<th>Dining</th>
<th>Kitchen</th>
<th>Bedroom</th>
<th>Bathroom</th>
</tr>
</thead>
<tbody>
<tr>
<td>Living</td>
<td>1.0000</td>
<td>0.4704</td>
<td>0.2563</td>
<td>0.9911</td>
<td>0.8483</td>
</tr>
<tr>
<td>Dining</td>
<td>0.4704</td>
<td>1.0000</td>
<td>0.9736</td>
<td>0.5835</td>
<td>-0.0683</td>
</tr>
<tr>
<td>Kitchen</td>
<td>0.2563</td>
<td>0.9736</td>
<td>1.0000</td>
<td>0.3826</td>
<td>-0.2944</td>
</tr>
<tr>
<td>Bedroom</td>
<td>0.9911</td>
<td>0.5835</td>
<td>0.3826</td>
<td>1.0000</td>
<td>0.7703</td>
</tr>
<tr>
<td>Bathroom</td>
<td>0.8483</td>
<td>-0.0683</td>
<td>-0.2944</td>
<td>0.7703</td>
<td>1.0000</td>
</tr>
</tbody>
</table>

Figure 6.2: Cosine measurements between spaces for house layout problem

Note how REIFORM infers implicit constraints from explicit ones, so that even in cases where the model is under or over constrained a topology may be generated. Figure 6.1 shows that a local constraint between the bedroom and the bathroom is undefined or neutral, i.e. score 0. The relationship between bathroom and dining is also undefined or neutral, i.e. score 0. From a general understanding that arises from having lived in a house, one may expect that the bathroom should be close to the bedroom. No such preference exists between the bathroom and the dining room (assuming that a sink in the kitchen or a wash area exists). Now, note from Figure 6.1 that even though both relationships were 0 in the occurrence matrix, the cosine measurement between bedroom and bathroom is high at 0.7703 in the $k$-reduced space, while the cosine measurement between the bathroom and dining space is negative at -0.0683. This is not surprising because REIFORM uncovers implicit constraints and relationships deriving from explicit ones in the problem representation. While both bedroom and bathroom share a high explicit relationship with the living space, causing their mutual cosine scores to go up, the bathroom and dining space do not share common high relationships with any other common spaces. This causes their mutual cosine measurement to go down.

Figure 6.3 shows the topology generated by the method, using $k=2$. Till this point, no cosine threshold is used, but the discriminatory power of the cosine measurements is used. The line thickness and color in the bubble diagram shows how desirable the connection is between the two spaces.

Based on this bubble diagram, the designer can now explore a range of topologies by varying the considered cosine threshold. Figure 6.4 shows a topology generated by using 0.4 as the cosine threshold, and a possible geometrical arrangement (schematic only). All spaces with mutual cosine higher than 0.4 are connected to each other (a red line shows a door or passage, i.e. a connection), and all spaces with mutual cosine lower than 0.4 are not connected to each other.

The cosine threshold may be varied for observing other topologies. Note that all such topologies will be feasible because the method will never allow a connectivity relationship to
be introduced between two spaces that share relatively low connection strength in the original matrix before allowing a connection between two spaces that have a higher connection strength in the original matrix. For example, note from Figures 6.2 and 6.3 that the connection between the living room and the bedroom is strong at 0.9911, and will be interpreted as a connection even if a cosine threshold of 0.9 is used.

**Figure 6.3: Topology solution for house layout problem**

REIFORM generates feasible solutions, and allows the designer to explore a range of topologies resulting from varying the matrix entries and the cosine thresholds. For problems with aesthetic criteria, (as in the “cultural” choice of client example presented above), generating a set of interesting feasible solutions may be more beneficial than generating the “optimal” solution. Further, in many types of problems (architectural design problems are a good example of this), the constraint set is a mixture of hard and soft constraints. Connections between some spaces must exist. Connections between some spaces are desirable. Connections between some spaces are undesirable. Connections between some spaces should not exist. The approach adopted in REIFORM identifies these in terms of range of varying connection “strengths”; a cosine threshold helps to identify a topology.

**Figure 6.4: Topology with cosine threshold = 0.4, k = 2, and a possible layout**
6.2.2 Floor layout problem

An extended version of this problem, also adapted from Michalek et al., is described as follows. A floor plan for an apartment complex is to be designed. The floor has 1 one-bedroom and 2 two-bedroom apartments. For a topology to be feasible, each apartment layout is constrained by the connectivity/adjacency constraints following the numbers presented in Figure 6.2, as presented in the previous section. An additional constraint is that the living space of each apartment must be connected to the common entryway for each floor that provides access to the apartments. Further, an obvious constraint is that internal spaces in each apartment should not be connected to any other internal space from any other apartment and the public entryway should provide the only common connection access to each of the apartments. Figure 6.5 shows an example occurrence matrix generated for the floor layout problem. The designer can experiment with other score values; the values used here are for demonstration. If the designer changes the constraint set, different scores may result.

<table>
<thead>
<tr>
<th></th>
<th>Entr</th>
<th>L1</th>
<th>D1</th>
<th>K1</th>
<th>Bed 1</th>
<th>Bat 1</th>
<th>L2</th>
<th>D2</th>
<th>K2</th>
<th>Bed 2</th>
<th>Bat 2</th>
<th>L3</th>
<th>D3</th>
<th>K3</th>
<th>Bed 31</th>
<th>Bed 32</th>
<th>Bat h 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entry</td>
<td>4</td>
<td>3</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>3</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>L1</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>D1</td>
<td>-1</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>K1</td>
<td>-1</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>Bath1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>4</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>L2</td>
<td>3</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>-1</td>
<td>-1</td>
<td>3</td>
<td>3</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>D2</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>K2</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>Bed21</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>Bed22</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>Bath2</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>4</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>L3</td>
<td>3</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>D3</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>K3</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bed31</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bed32</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bath3</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 6.5: Occurrence matrix for floor layout problem

The occurrence matrix shows that the housing layout problem has a special structure that leads to an obvious decomposition. The three apartments on the floor are internally contained (high intra-system connectivity) and connected to the others only through one common space (low inter-system connectivity). Due to this special nature of the housing layout problem, the initial occurrence matrix formulation shows an obvious decomposed structure (positive relationships between spaces in one apartment and negative between spaces of two different apartments). Figure 6.6 shows the results of the method applied onto the occurrence matrix.

Note again that a direct partitioning or clustering of the original occurrence matrix using only the explicit relationships will not be able to lead to this solution. Note from Figure 6.5 that there are 0 entries between bedrooms and bathrooms as well as dining rooms and bathrooms in the original matrix. However, in the solution shown in Figure 6.6 the relationships between bathrooms and bedrooms is high at 0.83 and 0.80, while the
relationships between dining rooms and bathrooms is low at 0.24 and 0.23. At the chosen cosine threshold of 0.6, this will lead to no connectivity between dining rooms and bathrooms, but a direct connectivity between bedrooms and bathrooms. This result occurs because of the manner in which the $k$ approximation induces implicit relationships using the explicit relationships in the occurrence matrix.

The table shows the relationship that only the living rooms of the three apartments have a cosine measurement of 0.45, 0.41 and 0.41 with the entryway, with the others all negative. Further, the internal spaces of the three blocks show positive relationships (high intra-system interaction) with negative relationships between the internal spaces of two different apartments (low inter-system relationships). Within the apartments (the colored blocks), low to high relationships exist between the spaces, grading the level of connectivity that each space must share with another. Figure 6.7 shows the topology generated for the full floor by considering a cosine threshold of 0.6. The connection between the public entryway space and living rooms of the apartments are considered positive even though their relationship is below 0.6, because of the obvious problem structure.

### 6.2.3 Discussion

Topology modeling problems have been described as combinatorial, multi-modal and highly constrained, with a discrete solution space (Michalek et al., 2002). Thus, the solution space must be searched globally. Exhaustive search is impractical for problems of any significant size due to combinatorial explosion. Stochastic search like genetic algorithms could be used, but they are computationally expensive. For example, the authors Michalek et al. (2002) report that for this apartment floor layout problem, the genetic algorithms they used produced consistent solutions when run for 20,000 generations with a 100 designs as each generation size. This implies $20 \times 10^6$ design evaluations were performed, which is a large number. This
implies significant computational expense and time. For this reason, they state that their method is more useful as a feasible design finder rather than a true optimizer.

**Figure 6.7: Topology solution for floor layout problem with cosine threshold = 0.6, k=7**

However, they note that in domains such as architectural design, it is much more useful to have a method to generate alternate feasible designs because optimal floor plan designs may or may not be acceptable based on other criteria such as aesthetics that cannot be encoded formally into the optimization model. Then, the designer can evaluate these alternatives for a final design solution. This indicates that for certain classes of problem modeling and formulation tasks, it is beneficial to have a reformulation method that can allow a designer to quickly define and redefine requirements and observe multiple solutions. In addition, the modeling method should be computationally efficient; otherwise, it does not fulfill the criteria of serving as an interactive, quick, design support mechanism.

Michalek et al. (2002) use the topology optimization module with a geometry optimization module. The geometry optimization module is not combinatorial and uses a formulation that employs fast and efficient gradient-based algorithms to optimize geometry. Therefore, computation time can be greatly reduced for a full optimization, if there was a fast reformulation tool that could generate good topology solutions interactively that could then be fed in to the geometry optimization module. REIFORM can provide the designer with a quick, interactive mechanism in the pre-modeling or modeling “bubble” diagram stage to do this.

REIFORM does not operate on rule based mathematical knowledge. Rather, it is based on a pattern based approach; it finds associational mappings between design concepts, and allows for exploration. It is a computationally efficient mechanism – doing the same problem for this thesis required a few minutes of programming time in Matlab and negligible
computation time in comparison to the evolutionary program used by Michalek. It allows the
designer to change the matrix entry values – an action that reformulates the solution space
that is to be searched for finding a solution. In addition, Figure 6.3 demonstrates that one of
the main strengths of REIFORM is that changing the cosine threshold levels will cause
multiple solution patterns to emerge – an action that allows the solution search process to
focus on multiple solutions. This implies that the same problem representation has the
capacity to be used to observe multiple solution alternatives.

6.3 Summary

This chapter presented the application of REIFORM for topology design as an example of a
constraint satisfaction design reformulation task. The interpretation and use of REIFORM was
a little different in this chapter as compared to the previous one. In Chapter 5, matrix
partitioning and clustering approaches were appended to REIFORM to identify
decomposition results. In this chapter, cosine measurements between “architectural spaces” in
the $k$-reduced approximations were used to assign topological connections between them.
Results from this chapter reinforce the claim that REIFORM can be used for all those types of
problem reformulation tasks that require the initial problem model to be “seen” as
decomposed into sub-structures in a general way. In this chapter, the aim was not a strict
decomposition in terms of partitions generated using event-episode occurrence patterns, but a
continuous general clustering in terms of varying cosine values that reflected constraints on
components’ connectivity. The occurrence matrix relations were explicit constraints on
components of a system. Finding the implicit relations using the explicit ones enabled
REIFORM to “pull together” all the highly connected components (for example, architectural
spaces that have positive/desirable connections and mutually reinforce each other in terms of
activities) and “push apart” all the weakly connected ones (for example, architectural spaces
that have undesirable connections). It is possible to see the conceptual similarity of the two
tasks, even though the final reformulation objective is different in the two cases.
Chapter 7
Identifying Variables, Functions and Design “Cases”

The variable comes to appear in its true light as purely a means of identifying and distinguishing the referential places in a [function] ... Such then is the cosmic burden borne by the humble variable. It is the locus of reification, hence of all ontology.

W V Quine, From Stimulus to Science

This chapter presents the application of REIFORM to aid selection of design variables and constraints and heuristically aid design “case” identification. A single objective formulation of a hydraulic cylinder design problem (Papalambros & Wilde, 2000) is used to demonstrate the method for these tasks.

7.1 Identifying linked variables and functions for problem modeling

One of the first steps in formulating or reformulating a design problem are decisions on which design elements to consider as decision variables, which ones to fix as parameters, and what functional relationships to consider between the chosen elements. Often, these decisions are based on previous experiences of a designer and the physics of the problem being modeled. It is likely that problems of the same class will share a similar set of design elements and functional relationships, even though different designers choose to model them in slightly or widely differing manners. That is, for a computational method, previous formulation samples will hold invariant patterns as well as multiple variant ones, which a computational method could uncover for use in a new formulation.

Even for a single formulation, design variables are semantically linked in terms of behavioral relationships (e.g., the hoop stress $s$ should be less than a maximum value $S$). When a designer considers one element as a variable, it becomes important to know what other variables, parameters or functions should be considered in conjunction. This semantic
knowledge is not always explicitly available in the original problem formulation. For example, is the hoop stress equation \( s - S <= 0 \) important enough to be considered as a constraint if \( i \) is being considered as a variable? Obviously, \( i \) does not occur in this equation, so how can one tell a priori?

For REIFORM, a single problem formulation statement is like a “training” set. It consists of episodes in which events occur in a certain pattern. That is, REIFORM is given a set of explicit relations. The 4 steps of REIFORM take these episodic occurrences and infer possible sets of semantic meaning implicit in these explicit relations. This knowledge can, then, be used to inform problem reformulation. Therefore, in the most primary way, given a certain problem formulation (as an encoding of a design “experience”), and a “query” (a variable, parameter or function), it can infer the other related variables, parameters or functions that are important in conjunction with the query. The steps are as follows:

1. Create the occurrence matrix (Step I) that captures the event-episode explicit mapping.
2. Use the SVD analysis (Step II) and dimensionality reduction (Step III) to re-represent all the design concepts (variables, parameters and functions) in a common space.
3. Choose a “query” variable, parameter, or function and fix a cosine threshold. That is, we wish to ask – given this variable, parameter or function, which other ones are linked to this query in the local set of events and episodes being considered.
4. Use the similarity measurement step (Step IV) to compute cosine distances, and return all the variables, parameters and functions that fall within the cosine threshold as design concepts linked to the query variable, parameter or function.

### 7.2 Linked variables lead to heuristic identification of design “cases”

Using the computations shown in the previous section, REIFORM can provide heuristic insights into design “case” identification (Papalambros & Wilde, 2000). In any problem formulation, the mathematical model contains a set of constraints. This model could be over or under-constrained, i.e. not well-formulated. Often, two constraints may not be critical for a problem model when considered together, but each could prove to be critical when considered individually (i.e. by removing the other from the model). Design “cases” are sets of constraints that, when considered together for a design problem, lead to a well-formulated problem model. One method for identifying these design “cases” (Papalambros & Wilde, 2000), i.e. sets of active or critical constraints for a problem model, is monotonicity analysis (Papalambros & Wilde, 2000). Monotonicity analysis is a problem-solving-by-reformulation method, where constraint activity information is used to reformulate the problem to a simpler form generally when a problem is over or under constrained. The method is often employed to discover design cases – sets of constraints that can be active or inactive, in order to reach
well-formulated models. A significant characteristic to note here is that this process of reformulation is actually similar to discovering previously unobserved implicit relationships between variables, parameters and constraints, that when made explicit, make solving the problem possible.

The 4 steps outlined in the previous section can aid the identification of design “cases” in a heuristic manner using a similar conceptual idea – *the inference of previously unobserved implicit relationships between variables, parameters and constraints*. Note that monotonicity analysis is a mathematically rigorous rule-based procedure and it can identify these cases without ambiguity, using formal definitions of criticality, activity, monotonicity and optimality. REIFORM can provide heuristic insights into design case groups in a “static” way, and cannot identify bounded cases as monotonicity analysis does, because it employs an unsupervised pattern recognition approach as opposed to a formal rule-based or knowledge-rich approach. For example, (as we show in the hydraulic cylinder example below), while it shows that there are groups of constraints that share implied and explicit relationships with each other, it cannot tell that these need to be active together as a case. However, monotonicity analysis is applicable only on problems where regions of monotonic behavior in constraints may be identified. It quickly turns into a very complex solution procedure for even a small-medium sized problem as the number of variables and constraints increase. REIFORM would be particularly suited for complex problems, where, if used in conjunction with monotonicity analysis, will be able to focus a designer’s attention on possible design cases to consider as semantically (implicitly or explicitly) related variables and functions. An added advantage of REIFORM is that it can be used to identify semantically related groups for analytic as well as non-analytic formulations, whether or not functional relationships are available, whether or not the functions are differentiable. This implies that for design problems where monotonicity is not present or the functions are not differentiable, REIFORM can still be used to provide the designer with some insight into how design “cases” may emerge.

The next section demonstrates the identification of linked variables and functions for problem modeling, and how this leads to heuristic design case identification by following the 4 steps outlined in this section.

7.3 **Hydraulic cylinder design problem**

The hydraulic cylinder design problem was chosen as a demonstration problem because it is widely referred to in the literature and serves as a test problem for many approaches (Michelena & Agogino, 1988; Papalambros & Wilde, 2000; Williams & Cagan, 1994). It has been formulated and solved in multiple ways. This thesis works with the commonly known
formulation of the problem – a single objective formulation (Papalambros & Wilde, 2000). Figure 7.1 shows the design problem and Figure 7.2 shows the problem statement. This problem has 5 design variables \( \{i, t, f, p, s\} \) and 4 design parameters \( \{T, F, P, S\} \). The model has 4 inequality constraints and 2 equality constraints; the objective is to minimize the cylinder diameter. REIFORM is demonstrated for the following tasks: (1) inference of linked design variables and constraints; and (2) heuristic design “case” identification.

![Hydraulic cylinder design (HCD) problem](image)

**Figure 7.1: Hydraulic cylinder design (HCD) problem**

<table>
<thead>
<tr>
<th>Design variables:</th>
<th>Design model:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal diameter: ( i )</td>
<td>Min ( i + 2t )</td>
</tr>
<tr>
<td>Wall thickness: ( t )</td>
<td>( g_1: t - T \geq 0 )</td>
</tr>
<tr>
<td>Output force: ( f )</td>
<td>( g_2: f - F \geq 0 )</td>
</tr>
<tr>
<td>Stress: ( s )</td>
<td>( g_3: p - P \leq 0 )</td>
</tr>
<tr>
<td>Pressure: ( p )</td>
<td>( g_4: s - S \leq 0 )</td>
</tr>
<tr>
<td>Design parameters:</td>
<td>( h_1: f = (\pi/4)i^2p )</td>
</tr>
<tr>
<td>Min Wall thickness: ( T )</td>
<td>( h_2: s = ip/2t )</td>
</tr>
<tr>
<td>Min Output force: ( F )</td>
<td></td>
</tr>
<tr>
<td>Max Stress: ( S )</td>
<td></td>
</tr>
<tr>
<td>Max Pressure: ( P )</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 7.2: Single objective formulation of the hydraulic cylinder problem**

7.3.1 Identifying linked variables and functions for problem modeling

For this problem, a query variable is considered, say internal diameter \( i \). We want to know which other variables and functions are explicitly or implicitly related to \( i \) such that they are important to consider in conjunction with \( i \). With a fixed \( k = 2 \) value, cosine measurements between \( i \) and all the other variables, parameters and functions are now computed in the \( k \)-reduced space. That is, for a particular \( k \) value, matrices \( X(2) \) and \( Z(2) \) are computed, and the row that represents \( i \) is selected. A cosine threshold is decided upon. All the variables, parameters and functions with cosine measurements with \( i \) that are higher than this threshold are returned as answers. Let us, for example, consider a cosine threshold of 0.7. Figure 7.3 is a graphical representation of the \( k=2 \) space, and the region within the arrows show which other elements, parameters and variables are returned as answers with the query variable set to \( i \) and the cosine threshold set to 0.7. This can be confirmed from Figure 7.4 that shows the cosine measurements that \( i \) has in the 2D space with all the other variables, parameters and functions (the first rows of matrices \( X(2) \) and \( Z(2) \)). Note that the query on \( i \) returns functions
$g_3: p-P\leq0$ and $g_4: s-S\leq0$. $i$ does not occur explicitly in either of these, but as will be seen in the next section, they are important functions to consider for variable $i$.

In the same way, any of the other variables, parameters or functions could be used as a “query”.

![Figure 7.3: Variables, parameters and functions returned for query variable $i$, $k=2$, cosine threshold = 0.7](image)

<table>
<thead>
<tr>
<th>Query</th>
<th>i</th>
<th>t</th>
<th>f</th>
<th>p</th>
<th>s</th>
<th>T</th>
<th>F</th>
<th>P</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>1.0000</td>
<td>0.7884</td>
<td>0.5126</td>
<td>0.9238</td>
<td>0.9347</td>
<td>0.2139</td>
<td>0.8224</td>
<td>0.5418</td>
<td></td>
</tr>
</tbody>
</table>

![Figure 7.4: Cosine measurements of $i$ with all other variables, parameters and functions, cosine threshold = 0.7, $k=2$](image)

<table>
<thead>
<tr>
<th>Query</th>
<th>$d$</th>
<th>$g_1$</th>
<th>$g_2$</th>
<th>$g_3$</th>
<th>$g_4$</th>
<th>$h_1$</th>
<th>$h_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0.8881</td>
<td>0.5500</td>
<td>0.3043</td>
<td>0.7722</td>
<td>0.7041</td>
<td>0.8216</td>
<td>0.9739</td>
</tr>
</tbody>
</table>

7.3.2 Identified linked variables and functions aid heuristic design “case” identification

Consider solving the single objective hydraulic cylinder design example problem using monotonicity analysis. This method is well documented on this problem by (Papalambros & Wilde, 2000) and (Michelea & Agogino, 1988). In the case of this example problem, there are 5 design variables, and 6 design constraints. The number of non-redundant, active constraints cannot exceed the number of design variables for a consistent solution to be found, revealing that there will be design “cases”. All the constraints cannot be active at the same time. There will be sets of active constraints, leading to different solutions. Papalambros and Wilde identify 3 design cases – stress-bound, pressure-bound, and thickness-bound. Their results show that for design variable $i$ (internal diameter) either constraints ($g_3$, ($g_2$, $h_1$)) or (($g_4$, $h_2$), ($g_2$, $h_1$)) will be active, and for variable $t$ (wall thickness) either constraints (($g_4$, $h_2$), ($g_2$, $h_1$)) or ($g_1$) will be conditionally critical. Figure 7.5 shows that the cosine measurements (or an equivalent K-means clustering) performed between function vectors in the 2D space identify similar conclusions through a purely syntactic analysis of the design
formulation. Observe that constraints \((g_2, h_1)\) and \((g_4, h_2)\) form distinct visual groups, with constraints \(g_3\) and \(g_1\) falling close to these two groups, a fact numerically confirmed by the cosine measurements. With a cosine threshold of 0.7, observe from Figure 7.6 that \((g_2, h_1)\) and \((g_4, h_2)\) both have high cosine measurements, showing that they occur as a semantic “group”. In addition, \(g_3\) shares a high relationship with \((g_2, h_1)\) (a design case for \(i\)) and shares a low relationship \((g_4, h_2)\) (another design case for \(i\)). Similar analysis can be made for wall thickness \(t\).

![Figure 7.5: Semantically related constraint groups identified by REIFORM, cosine threshold = 0.7, \(k=2\)](image)

<table>
<thead>
<tr>
<th>(d)</th>
<th>(g_1)</th>
<th>(g_2)</th>
<th>(g_3)</th>
<th>(g_4)</th>
<th>(h_1)</th>
<th>(h_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(d)</td>
<td>1.0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(g_1)</td>
<td>0.8724</td>
<td>1.0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(g_2)</td>
<td>-0.1674</td>
<td>-0.6280</td>
<td>1.0000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(g_3)</td>
<td>0.3936</td>
<td>-0.1060</td>
<td>0.8404</td>
<td>1.0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(g_4)</td>
<td>0.9518</td>
<td>0.9803</td>
<td>-0.4619</td>
<td>0.0925</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>(h_1)</td>
<td>0.4679</td>
<td>-0.0239</td>
<td>0.7930</td>
<td>0.9966</td>
<td>0.1741</td>
<td>1.0000</td>
</tr>
<tr>
<td>(h_2)</td>
<td>0.9692</td>
<td>0.7281</td>
<td>0.0806</td>
<td>0.0997</td>
<td>0.8468</td>
<td>0.6711</td>
</tr>
</tbody>
</table>

![Figure 7.6: Cosine measurements for groups shown in Figure 5.13](image)

Note again that REIFORM can only provide insights into design case groups, and cannot provide unambiguous cases as monotonicity analysis does, as the method structure operates only on the symbolic event-episode occurrence information and does not incorporate any “dynamic” activity analysis. For example, while it shows that there are groups of \((g_2, h_1)\) and \((g_4, h_2)\), it cannot tell that these need to be active together as a case. However, as discussed before, for problems where the required monotonicity or continuity conditions are not met, the approach suggested by REIFORM could be applied for some heuristic insight into such “cases”.

### 7.4 Summary

This chapter demonstrated the use of REIFORM for the problem reformulation tasks of identifying linked variables and constraints and heuristic identification of design cases. A
single-objective hydraulic cylinder problem was used as a demonstration example for these tasks. In problem reformulation, part of “learning” how to reformulate design problems involves using the episodic information already available in the current formulation. REIFORM uncovers such information by inferring previously unobserved implicit relationships in the event-episodic “training” set. This knowledge then informs problem reformulation.
Chapter 8
Heuristics for Parameter Selection

...everything we know is only some kind of approximation, because we know that we do not know all the laws as yet... The test of all knowledge is experiment...But what is the source of knowledge? Where do the laws that are to be tested come from? Experiment, itself, helps to produce these laws...also is needed imagination to create from these hints the great generalizations – to guess at the wonderful, simple, but very strange patterns beneath them all, and then to experiment to check again whether we have made the right guess.

Richard Feynman, Lectures on Physics I

Chapter 4 presented the REIFORM method. Chapters 5, 6 and 7 demonstrated the application of REIFORM for various types of problem reformulation tasks. It has been shown that REIFORM retrieves explicit and implicit semantic design information from the syntax of design representation. The information that it retrieves can be used to reformulate the problem. However, the retrieval of information is based upon several parameters that a user can change. Since this was left as an open question in the previous chapters, this chapter presents heuristics to guide the selection of values for these parameters. These parameters are: (1) the number of singular values $k$ retained in the dimensionality reduction step that produces an approximation of the occurrence matrix; and, (2) the cosine threshold used in the clustering or similarity measurement step and the number of clusters used in the K-means algorithm.

These heuristics address the question, “How does a designer decide what $k$ values, cosine thresholds, or number of clusters to choose such that it leads to “good” reformulations?”

8.1 Choosing the number of dimensions $k$

Recall again the 4 steps of REIFORM. The occurrence matrix captures explicit relationships. SVD analysis re-represents the explicit relationships into distributed, implied relationships. The singular values preserve the association “strength” information in a decreasing order of
magnitude (since they are arranged in a decreasing order of magnitude, and each $k$ level approximation is the best rank 1 approximation to the original matrix in the linear least squares sense). In effect, the $k$-reduced approximations produced by the first few largest singular values can induce implicit relationships between the design concepts that cannot be derived from the original occurrence matrix. Note that this interpretation is different from the conventional approximation “error” concept – in the conventional interpretation (as in say the digital image processing examples in Chapter 2) the aim is to employ the redundancy of information in the original matrix to find an optimal $k$ value such that a $k$ approximation is able to return the same information as the original rank $r$ matrix, leading to data compression. However, for this work, along with the explicit information, we also need the implicit information that is induced. The interpretation is that some of the “error” is actually useful, because it induces these implicit relationships, using the explicit information, that are actually useful for reformulation tasks. The task is to distinguish which of these implied levels of abstraction, i.e. $k$ values, are “implied patterns” and which of these are “noise”.

Finally, recall that as the $k$ values increase, the $k$ approximations approach the limits set by the entries of the original occurrence matrix. That is, there is a continuous gradation of interaction strengths between the most “implicit” end (largest $k$ values) and the most “explicit” end (the full rank $k = r$ approximation). Thus, as the $k$ value increases, the information returned by the approximated matrix comes closer and closer to the explicit information contained in the occurrence matrix. Figure 8.1 shows a visual interpretation.

Figure 8.1: Visual interpretation of induced, implied patterns for the $k$-reductions

The first few singular values $k = 2$ to $a$ will return implicit information. As the $k$ value increases beyond $a$, then all entries in the approximated $k$-reduced matrices would start going to the limit values set by the explicit occurrence information in the occurrence matrix. For example, if the mapping in the occurrence matrix is binary, as $k$ values increase, the values in the approximated matrices would start coming closer and closer to the original 1 or 0 explicit relations. This will cause the cosine values to measure associative distances between events and episodes in the $k$-reduced $US$ and $SV^T$ spaces as closer and closer to those in the $k = r$ full rank $US$ and $SV^T$ spaces. As shown in Section 4.3.4, if the cosine threshold parameter is fixed for each $k$, at some $k = a$, and $k \leq r$ (where $r$ is the rank of the occurrence matrix), the
approximation will start to return only the explicit association information as contained in the occurrence matrix. Therefore, as a first rule, to observe implicit associative information that leads to multiple reformulation decisions, use $k = 2$ to $a$. Beyond $a$, the approximation will return only explicit associative information, as can be read off directly from the occurrence matrix, or cosine measurements between $UV$ and $SV^T$ vectors in $k = r$ space. All reformulation tasks demonstrated in this thesis are based on the use of this implicit information. Therefore, the search for well-formed reformulations is limited within this range of $k = 2$ to $a$.

As the main heuristic rule, out of all the $2$ to $a$ approximations, a good value of $k$ is one that would help to correctly alter a not well-formulated problem into well-formulated ones. The method to identify a well-formulated problem is as follows:

1. Apply REIFORM for $k = 2$ to $r$. Identify the value $a$ beyond which no implicit information is returned through a parametric study that increases the $k$ by steps.

2. Fix a cosine threshold, and use the $US$ and $SV^T$ approximations in the $k$-reduced spaces from $2$ to $a$ to study the reformulations returned for each of the $k$ values. If there is a well-formed reformulation, the method will return this.

For example, in the problem decomposition task, if a decomposition exists, then a “good” $k$ value would lead to the identification of block matrices within the larger matrix. If a block matrix is not identified at a $k$ value, then either no solution exists (for example, the problem could be very highly coupled, such that no block matrices can be found for any cosine threshold); or the $k$ value a “good” value. In decomposition tasks, the main idea was to limit the search for $k$ values to those that may maximize intra-sub-problem interaction, and minimize inter-sub-problem interactions in terms of induced distances in the $k$-reduced space.

Similarly, for the topology modeling task, a “good” $k$ value would lead to a topology that satisfies the explicit constraints. A parametric study of the large apartment layout problem from Chapter 6 Section 8.1.1 shows how some values do not lead to “good” topology solutions while others do.

This is a heuristic. Identifying the optimal range of values of $k$ (where “optimal” implies that range of $k$ values that leads to well-formed reformulations) the algorithm will have to be developed. These ideas for theoretical and formal method development that focus on identifying the optimal $k$ are presented as future work in Chapter 10. However, this is more in line with the idea of providing a formal proof that the method actually returns the “correct” reformulations. On the basis of the work presented here, the conjecture is that the results obtained from the method will remain the same, even with a formal proof.

Next, to make the heuristic presented here stronger, two patterns in which $k$ identification can occur are presented, with the help of two parametric studies that bring out these two different patterns.
8.1.1 Values of \(k\) close to 2 do not lead to well-formulated solutions

As a demonstration for this first pattern, consider the topology design (large apartment, floor layout problem) example from Section 6.2.2. Sometimes, a too low or too high \(k\) value, i.e. close to 2 or close to \(a\), will not lead to a well formulated problem. Recall from Section 6.2.2 that the apartment layout topology problem was a problem that had an inherent disjoint structure. The explicit information contained in the occurrence matrix showed three block matrices for the three apartments (Figure 6.5). Recall also, that a \(k = 7\) value was used to demonstrate the topology solution. Figure 8.2 shows the \(k = 2\) approximation.

![Figure 8.2: Cosine measurements between spaces for the floor layout problem, \(k=2\)](image)

This shows that although the solution is a “correct” one, (three block matrices are identified) it does not lead to a well-formulated problem because it does not have enough discriminatory power to show the intra-apartment space relationships. The \(k = 2\) approximation correctly returns the most important associative pattern – that the three apartments form three “sub-systems” with high intra-system interaction, and low inter-system interaction, but this is not enough to lead to a topology solution. This approximation fails to discriminate between the internal apartment space relationships. The next most important associative patterns are needed to uncover these relationships. Therefore, from \(k = 4\) onwards, different patterns of association between the internal spaces provides well-formed solutions. A \(k = 7\) approximation was used for demonstration; the \(k = 4\) to 10 range gave feasible topology solutions. This shows that the relevance of the answers returned is low at \(k = 2\), then starts to increase, and then beyond a certain \(k\) value starts to decrease again, as the approximations go closer to the occurrence matrix.

8.1.2 Values of \(k\) from 2 to \(a\) return well-formulated solutions with gradually decreasing relevance

For this second pattern, consider the aeroengine problem, which is an example of a complex system. Complex systems are characterized by either large problem sizes in terms of the number of variables/ system components and relationships, or they are characterized by
strong interactions and couplings between them. The aeroengine problem shows both these characteristics. Figure 8.3 shows the components that REIFORM returns as answers for the query component 1 as the \(k\) value is varied from 2 to 54 (i.e. number of components sharing cosine measurements higher than the cosine threshold with Component 1: Fan Containment Case), and the cosine threshold set to 0.5. That is, the \(k\) value is varied, the cosine threshold and query variable are fixed, and the results plotted. The x-axis shows that the \(k\) value is varied from 2 to 54. For each \(k\), components that show cosine measurement values higher than 0.5 with component 1 are counted and plotted. The numbers of components returned are plotted in terms of sub-systems as well as the total number returned. Figure 8.3 confirms that as the \(k\) value is increased, the answers returned approach closer and closer to the limits set by the \(k = 54\), i.e. the original occurrence matrix.

![Component 1 - other sub-system component relationships](image)

**Figure 8.3:** Parametric study of \(k\) on the aeroengine problem, analysis for component 1 as “query”

Note the following observations from the charts:

1. The first column shows the explicit relationships – the number of components with which component 1 has a direct relationship with, as specified in the occurrence matrix. When the \(k\) value is varied from 2 to 54, at \(k = 54\), the USV\(^T\) decomposition will generate the original occurrence matrix again, so the number of explicit relationships in the occurrence matrix, and the number of relationships returned by cosine measurements at \(k=54\) should be
the same. Figure 8.3 confirms this – the first (occurrence matrix) and the last set \( (k = 54) \) of observations is the same in all cases.

2. At \( k = 2 \), the highest number of components are returned, with the number gradually decreasing. At around \( k = 15 – 20 \), the values stabilize and become the same as the explicit occurrences in the occurrence matrix. This implies that all the implicit relationships that are captured lie in the region \( k = 2 \) to 10. This also implies that between \( k = 15 \) to 20, the truncated \( k \)-reduced approximation matrix starts to return the occurrence matrix relationships, and beyond \( k = 15 \) up to \( k = 54 \), the original occurrence matrix information is returned.

3. Finally, recall from Chapter 6 that at \( k = 2 \) REIFORM returned patterns that could correctly classify the modular and integrative systems. Thus, even though the problem size and coupling is larger in this case (as compared to the topology layout case), the \( k = 2 \) approximation returns a well-formulated decision in this case, and is unable to in the topology case. Further, between \( k = 2 \) to 15, different associative patterns show multiple decisions on the “degree” of modularity for each component, i.e. to what extent a system is modular or integrative. Though the main classification decision on the modular and integrative systems does not change, varying \( k \) from 2 to 10 shows the varying degrees to which each system can be considered modular or integrative.

To generalize this discussion, parametric studies were performed for each of the example problems. There were two patterns that came out of the experiments performed in this thesis. Figure 8.4(a) shows the first pattern (as exemplified by the aeroengine problem) – as the \( k \) value increases, the number of “correct” answers that lead to well formulated problems reduces gradually, with the lowest \( k \) values providing the best information. That is, the first few \( k \) values led to well reformulated problems. Empirically, this pattern came from cases that have a moderate to high complex structure – large problem size or high couplings.

Figure 8.4(b) shows the second pattern (as exemplified by the topology design example) – as the \( k \) value increases, the number of “correct” answers that lead to well formulated problems first increases, and then decreases again. That is, the first few \( k \) values failed to lead to good solutions; as the \( k \) value increases, the information returned leads to good solutions; beyond a certain range the information returned again does not lead to good solutions, because the matrix entries start to come closer to the explicit occurrence information. This pattern came from cases that, like the topology modeling case, already have an obvious decomposed structure to start with. Note that these are schematic patterns only, and many more experiments would have to be conducted to plot these from a statistical perspective.
It is obvious that when \( k \) equals the original number of dimensions (the rank of matrix \( A \)), then the semantic groups return only the explicit information contained in the original data matrix. With reduced dimensionality representations, the algorithm captures induced semantic relationships that are not explicitly observable from the original matrix. The significance of the method lies in its ability to use the symbolic syntax of problems to provide help in identifying well-formulated design optimization problems in terms of “good” groupings of variables and constraints. Choosing good \( k \) values leads to REIFORM returning multiple associative patterns at varying abstraction levels, which become the basis for a reformulation decision.

### 8.2 Choosing cosine thresholds and number of clusters in K-means

The cosine measurement is a measure of the interaction strength between events and episodes in any \( k \) space. Choosing a single “good” cosine threshold value and number of clusters in the K-means is the most important for decomposition type tasks. For other tasks, for example the topology examples, the general interpretation of the cosine threshold (or the number of clusters) is increasing or decreasing levels of interaction strength, where choosing different levels of coupling strength would lead to different reformulations. If a high threshold is chosen, fewer numbers of related events and episodes will be returned. If this threshold is relaxed, a higher number of related events and episodes will be returned.

Note that the numerical value of the cosine may not be important for deciding the threshold. That is, there is no standard that establishes whether 0.5 or 0.9 is a good threshold. It is the difference suggested by the values that are the basis for choosing it.

Therefore, the heuristic for choosing a good cosine threshold is that it should bring out the structure contained in the relative relationships at a single \( k \) level. For any \( k \) level, the heuristic rule is that a good cosine threshold is one that leads to the identification of a clear structure in the relationships that constitute a well-formulated problem. Thus, to identify a cosine threshold, follow these steps:

(i) find out the lowest and the highest cosine values in the matrix that is to be partitioned;
(ii) choose the highest cosine value, and use it to identify block matrices;

(iii) if no block matrix is identified, lower the cosine threshold value to the next highest value and repeat step (ii); (for identifying by how much the threshold should be reduced this thesis employed the average range of values as a guide. For example, a lowering of the threshold by 0.05 is no use if it is obvious that the differences between values is more to the order of 0.1)

(iv) if for any cosine threshold there is a well-formulated solution, i.e. block matrices are identified, then this is a well-formulated solution or the $k$ value has to changed and the cosine threshold reapplied; else,

(v) there is no well-formulated solution, i.e. decomposable structure in the problem as stated in its current form. As shown in Chapter 5, the method will identify the block matrices if there is a “decomposable” structure in the problem, because such a structure will result in cosine values that correspond to high intra-sub-system and low inter-sub-system interactions.

For problem decomposition tasks, a cosine threshold should be chosen that identifies the block matrices unambiguously. If the cosine threshold is say 0.9, and block matrices are identified on this basis, but the cosine values surrounding the block matrices show measurements such 0.8, then this may not be a good threshold. If the value is 0.9, and the surrounding values are 0.5, or -0.5, then this is a better threshold. This is roughly a kind of sensitivity analysis.

Depending on the problem structure, the cosine values vary across the $k$ approximations. The threshold value could be kept the same across varying $k$ levels, or a new one could be chosen for each $k$ level. For example, in the aeroengine problem, a value of 0.5 was chosen and kept constant in order to perform the analysis presented in the previous section, because this led to consistent answers being returned. In other cases, the general heuristic followed was to keep the difference between the cosine values observed at a single $k$ level as the basis for choosing the numerical value of the threshold.

The choice of a cosine threshold is also affected by the problem structure itself. If the problem does not contain any decomposable structure, then it will not be possible to identify a cosine threshold. The choice of a good cosine threshold rests on the existence of a good structure in the matrix.

Choosing the number of clusters in the K-means algorithm applies mainly to problem decomposition type tasks. If the cosine measurement matrix shows a decomposable structure, then the number of clusters can be heuristically chosen to be around the expected number of block matrices being identified and vice versa. The main purpose of having a clustering algorithm with a partitioning algorithm was to use them simultaneously to co-validate the results produced by one using the other. This was important to ensure consistency, because
the method employs heuristics to identify solutions. If the method is extended to provide a formal, theoretical proof of the best $k$ values and cosine thresholds (Chapter 10) then either a partitioning or a clustering approach could be used, without loss of validity.

8.3 Summary

This chapter presented heuristics for choosing “good” values for the number of dimensions retained $k$ in the dimensionality reduction step, and cosine thresholds and number of cluster in the K-means algorithm in the unsupervised clustering step. The “good” values will depend significantly on the problem domain, formulation and complexity. Therefore, for any problem, an in-depth analysis by varying all the parameters and use of the heuristics presented here should lead to useful reformulation decisions.
Chapter 9

Summary of Results

Consider what effects, that might conceivably have practical bearings, we conceive the object of our conception to have. Then, the whole of our conception of these effects is the whole of our conception of the object.

C S Pierce, How to make our ideas clear

This chapter assesses REIFORM and its performance against the six established behavior criteria (Chapters 2 and 3). Using this analysis, this chapter summarizes, in the form of four postulates, the relationships between the syntax of symbolic-mathematical design representations, the explicit and implicit semantic meaning contained in them, and computable reification mechanisms that connect the two as scoped and suggested by the performance of REIFORM.

9.1 Assessment of REIFORM’s performance against the 6 behavior criteria

This section shows the six behavior criteria of REIFORM and an assessment of the way in which these criteria are fulfilled by the various steps of REIFORM.

9.1.1 C1: The method should be knowledge-lean.

Behavior of REIFORM:
REIFORM Step 1: Generation of an occurrence matrix from analytical or non-analytical representations.

Assessment of behavior(s) to fulfill criterion:
The occurrence matrix generation requires no knowledge other than the symbolic event-episodic knowledge to perform the tasks demonstrated in Chapters 4, 5, 6, and 7. Since only the knowledge of the structure and behavior of a design as available in a single formulation is required, and no additional design domain knowledge needs to be provided to the method, it is claimed that the method is knowledge-lean.
9.1.2 C2: The method should be training-lean and should use the knowledge acquired in a design experience to act in the same experience.

**Behavior of REIFORM:**

REIFORM Steps I, II and III: These steps take the knowledge provided in one design formulation, and through the occurrence matrix generation, SVD analysis and dimensionality reduction, induce approximations of associative patterns from which decisions to reformulate the same problem representation can be made.

**Assessment of behavior(s) against criterion:**

Each single formulation example serves as a complete “training” set as it contains multiple “samples” – as episodes (functional relations) formed by events (variables). In a way, REIFORM performs unsupervised inductive inference. By using the functional relations as “training episodes”, it can acquire explicit and implicit “rules” about how variables and functions relate to each other. While supervised learning methods, such as decision tree induction, use labeled sets of training data to find “rules” for classification, REIFORM does so in an unsupervised manner, finding the “rules” from the structure inherent in the data. The knowledge induced from a single formulation is then used to inform reformulation of this problem. Therefore, REIFORM is training-lean.

9.1.3 C3: The method should allow heuristic problem exploration by inferring multiple interpretations from a single problem representation, and should be general, i.e., applicable across problems of varying complexity, domains and representational forms.

**Behavior of REIFORM:**

REIFORM Step I: As long as a design problem (from any design domain using any representational form) can be converted into an occurrence matrix form that captures mappings between two or one types of design representation types (such as events-episodes as used in this thesis), REIFORM can be applied onto it.

REIFORM Steps III and IV: These steps allow multiple interpretations (that have been shown to be valid, well-formed reformulations) to be derived from the same problem representation using (a) reduced dimension approximations $k = 2$ to $a$ and (b) varying cosine thresholds for a single $k$ as presented in Chapter 8. For a certain cosine threshold, dimensions $k = a$ to $r$ (where $r$ is the rank of the occurrence matrix) return the same information as the original matrix and therefore, no implicit information can be retrieved in the range of $k = a$ to $r$.

**Assessment of behavior(s) against criterion:**

Since problems of varying complexity from different domains and representational forms can be converted into a single common representational format of the occurrence matrix, the method is claimed to be general.
Since it employs this single common representation to infer multiple reformulation decisions in a variety of reformulation tasks, by using the $k$ values and cosine thresholds, it is claimed to be a heuristic, exploratory method. It has been shown that these multiple reformulation decisions are valid decisions leading to well formed reformulations of the original problem form. Note that the heuristically guided search for the $k$ values and the cosine thresholds should be considered as “tunable” parameters in an empirical approach to problem reformulation. Having a tunable parameter in this method is no different than choosing a cost function for an artificial neural network depending on the learning task, or the “split points” in a decision tree until satisfactory categorization results are achieved.

9.1.4 **C4:** The method should be able to model the distributed map of association patterns between symbols in a design experience in a dynamic manner.

**Behavior of REIFORM:**
REIFORM Step II: If a single matrix entry in the occurrence matrix is changed, then the SVD computation changes all the entries in the resulting decomposition.

**Assessment of behavior(s) against criterion:**
The nature of SVD is such that it produces a unique diagonalization of the occurrence matrix (orthonormal-diagonal-orthonormal) in re-representing the association strength between sets of symbols in terms of a “distance” in continuous space. If any one entry in this matrix changes, say from a 0 to a 1, then this has the potential to cause a change in all of these orthonormal and diagonal vectors. In effect, a single change re-arranges the positions of the all the elements in this re-represented space. Local, explicit changes introduce global, implicit changes. The conjecture here is that “meaning” is a dynamic higher order phenomenon rather than an explicit one-to-one mapping between symbol and meaning. If the symbol mapping changes in a local context, then this dynamically induces global changes in the way every symbol relates to every other symbol. The “same symbol becomes a different symbol”. Therefore, it is claimed that the method models the distributed map of association patterns between symbols in a design experience in a dynamic manner.

9.1.5 **C5:** The method should be able to retrieve and infer not just the explicit one-to-one mapping between symbols, but also the implicit meanings.

**Behavior of REIFORM:**
REIFORM Step II: A discrete, interaction strength matrix is converted into a continuous, distance based representation in Euclidean space, where distance is proportional to interaction strength.
REIFORM Step III: Implicit relationships are induced in lower dimensions using explicit relationships.
Assessment of behavior(s) against criterion:
The occurrence matrix captures the explicit design relationships. The above steps use this explicit local information to induce implied global information that may not be observed directly from the original matrix. This implied global induced information is used to inform the reformulation tasks, which, in turn, cannot be performed without using these implicit relations. The reduced dimensionality approximations in Step III have been shown to retain the invariant aspects (the strong explicit relations) in the original formulation. At the same time, it also infers the implied relations that are not directly available in the original occurrence matrix. Therefore, it is claimed that the method is able to retrieve and infer not just the explicit one-to-one mapping between symbols, but also the implicit structural and behavioral relations in a design formulation.

9.1.6 C6: The method should be able to use the episodic relationships in a design representation as the basis for reformulating a design and constructing alternate design representations.

Performance of REIFORM:
REIFORM Step I: The occurrence matrix contains the current event-episodic relationships
REIFORM Steps II and III: These steps re-represent the current relationships and induce other implied relationships.
REIFORM Step IV: Inference and clustering of these implied structural and behavioral relationships and its use on the same problem representation causes the construction of an alternate design representation.

Assessment of behavior(s) against criterion:
In design reformulation, the information available in an immediately preceding set of relationships is used to inform the next iteration of reformulation. Through the use of the four steps, REIFORM is able to demonstrate this behavior. It uses the current explicit relational information to induce the knowledge that is used to reformulate this same representation.

9.2 Summary postulates
Drawing from the assessment and analysis presented above, this section presents some summary design theory postulates.

Postulate 1: A symbolic-mathematical design formulation embeds the semantic meaning of a design object in explicit and implicit ways.
From a purely syntactic perspective, the mathematical model of a design work contains symbols and relationships between symbols. From a semantic perspective, this model takes a certain form because a designer intends to encode some “meaning” into each symbol and
relationships between symbols. This meaning either has to do with the physics of the system, or has to do with the subjective choices made by the designer. For example, while the choice of material may be a subjective choice made by the designer, the ways in which the material behaves is dependent on the laws of physics. Further, there are physics “rules” for modeling material behavior, but the decision to model a “rule” explicitly is also a choice exercised by the designer. Consider an example. The choice of material for a building envelope is a subjective choice available to the designer. This choice would be dependent, among other criteria, on the heat gain characteristics of each material and the resulting heat lag cycle it produces inside the building; these behaviors derive from the laws of physics. However, whether to model the physics of heat gain in terms of wall thickness as a variable or wall surface treatment as a variable is a choice made by the designer. That is, modeling assumptions include structural choices as well as choices on which physical behaviors to model and how to model them.

The mathematical model is an abstraction – an intentional decision to represent some explicit meaning using a set of symbols and some relationships. Because the mathematical model is an abstraction, it follows that there are other relationships that are plausible between the same set of symbols in the same context, but not represented explicitly. Further, explicit design relations may be a mixture of functional, spatial, or incidental relations. In general, when we model, we try to decompose them and model a problem appropriately. In reality, all relations exist all at the same time. SVD basically “remakes” these relations without deference to maintaining a separation, which probably existed in the original formulation for cognitive load reasons more than anything else.

The performance of REIFORM shows that some part of such plausible but unrepresented meaning is implicitly embedded in the model itself through associative relationships between symbols, and can be derived from the explicitly represented meaning. This implicitly embedded meaning is not evident or directly observable from the representation. For example, the topology modeling examples in architectural design show that a set of explicit, desirable or undesirable spatial relationships automatically induces a set of implicit relationships between spaces for which no explicit relationship has been specified. Because a few spatial relationships are explicitly specified, the others also become mutually constrained in an implicit way. It is usually not possible to define a full set of explicit constraints for such models. In any case, defining a full set of constraints for a complex problem may result in an over-constrained model and conflicts between constraints and inconsistency in solutions. The performance of REIFORM shows that even if a model is incomplete, i.e., all possible interactions and relationships are not described, REIFORM can induce these using the ones that are explicitly identified.
Postulate 1 is useful for problem reformulation because it allows a single representation to simultaneously encode multiple layers of meaning – the explicit layer and the implicit layers deriving from these explicit layers. Any of these implicit layers of meaning could be made explicit, thereby changing the model representation. Therefore, the capability of REIFORM to exploit a single problem representation to encode multiple meanings provides the basis for many reformulation decisions to be inferred from a single problem representation.

Postulate 2:

Explicit meaning arises from the locally represented mathematical mapping between two symbols. Implicit meaning arises from the global, contextual, non-explicitly represented associative relationships that a symbol has with all the other symbols; multiple global associative relationships (implicit meaning) derive from the local explicit relationships (explicit meaning).

A mathematical rule connects symbols in an explicit manner. There is a one to one mapping between the symbol and its meaning. There is also an explicit mapping between a symbolic expression (function) and the behavior that it describes. Then, how do the symbols and relationships encode implicit meaning?

REIFORM models the design reformulation problem using a pattern recognition and extraction approach (SVD). Steps II and III (SVD and dimensionality reduction, respectively) show that global implicit meaning can be revealed from the associative patterns that symbols share with each other through local occurrences with other symbols, regardless of whether they share an explicit relationships with all. That is, REIFORM has the capacity to model all potential relationships between all symbols inherent in the explicit formulation, based on the explicit relationships that some symbols share with some others.

Postulate 2 is useful for problem reformulation because it suggests that, in addition to a rule-based, “symbolic AI” approach based on explicit local symbolic manipulation, some reformulation tasks can also be based upon global pattern recognition and extraction (for example the inference of interaction/coupling strengths between events and episodes as shown by REIFORM). Both approaches will reformulate the syntactic structure of a problem and therefore change the semantic meaning the model encodes. The difference is in the way it is done. For example, a process such as monotonicity analysis analyzes the local occurrence relationships of symbols in constraints in an explicit and systematic manner, explicitly reducing model dimensions by identifying criticality, i.e. a constraint that is identified as active at the optimum is changed to an equality and used to reduce the number of variables in a model. However, it requires monotonic and differentiable smooth functions to ensure its application. In cases where these conditions are not met, heuristic model reduction could still
be performed using the approach suggested by REIFORM, that is, by identifying the relationships shared between variables and constraints in a global sense and the “clusters” of linked variables and constraints that affect each others’ activity.

**Postulate 3:**

*Explicit and implicit semantic meanings of a design can be acquired by inducing the correct levels of abstraction to view the associative relationships between symbols.*

Explicit and implicit meanings lie embedded in the design problem representation. In what way are they connected and how can this relationship between make explicit such that implicit meaning can be measured? At one end lies the explicit representation. This level of abstraction states that only the relationships that are described in the design model exist; these define the only possible relationships that can exist between symbols. At the other end lies the “coarsest” level of abstraction, which states that all symbols are related, positively or negatively, to each other on account of the fact that they occur together in the same design experience. The local relationships between them can produce a global pattern of relationships between them in terms of how they mutually affect each other. Between these two limits exists a range of continuously varying possibilities that describe the different degrees to which symbols are related to each other.

REIFORM interprets that the “coarse” end, at $k=2$, shows the maximum variance in how two symbols could be related in a design experience. The other extreme end is the optimal approximation, $k = \text{rank of the occurrence matrix}$. This level provides exactly the same meaning as in the explicit representation. Between these two ends lies a continuously varying range of possible implicit relationships between the symbol set.

Similarly, at any $k$ level, the cosine threshold (or the number of clusters in the K-means algorithm) provides another abstraction mechanism. A lower (“relaxed”) threshold implies an interpretation that returns a higher number of design events and episodes as coupled to each other. A higher (“strict”) threshold does the opposite – it returns relatively fewer numbers of design events and episodes as coupled to each other.

Using these two mechanisms, it was possible to search for a range of well-formed reformulations of the original problem.

*Postulate 3 is useful for design reformulation because,* for reformulation to occur, the same “symbol” abstraction needs to be viewed differently. The fixed relationships that one symbol has with another need to be “unfixed”. Reformulation of design models requires processes of alternating between fixing and unfixing these relations between symbols. Fixing them gives the power of mathematical analysis; the basis for analysis is the model with explicit representations of relations. Unfixing it gives the power of synthesis; the basis for
synthesis is that potential and flexible meanings need to arise for the reformulation to happen. By synthesis in this context is meant any reformulation that results in changing the space of possible solutions and possible reformulations encoded by the model.

Postulate 4:

_Some design reformulation tasks can be modeled as processes that use the implicit meaning globally acquired from local explicit relationships to change these explicit relationships._

Design reformulation is a syntactic act that involves changing symbols and their relationships in the model. On the semantic plane, this syntactic act signifies that the way in which a designer understands the “meaning” of the design object has changed, and this is reflected in the syntactic changes. Often the observation of the formulation itself leads to possible reformulations.

The behavior of REIFORM shows that capturing the implicit potential meanings and using these to change the syntactic formulation can reformulate the design object being modeled.

_Postulate 4 is useful for design reformulation because it presents an alternate perspective for developing reformulation algorithms. Design is a higher-order cognitive activity involving symbols in large part. From a pattern based view, all symbols share some semantic relation on account of the fact that they occur in the same design experience. Semantic meanings are polysemous and contingent upon enacted meanings (the meaning that is possible given experiences and current perception). Explicit meaning is expressed through formal symbolic relationships. Implicit meaning is generated through the associative relationships that a symbol shares with others that derive from the explicit meaning, but are not directly observable from the sparse representation. A single symbol can influence the meaning of other symbols based on the statistical pattern of their occurrence. What REIFORM does is calculate the variation of the occurrence of a symbol to show how it affects the expression of that symbol and the functioning of that symbol within a larger symbolic context in producing semantic meaning. This postulate suggests that the development of design computation algorithms can be based on this statistical unsupervised pattern recognition and extraction view for some design tasks that have conventionally been solved using a supervised, rule-based, symbolic AI approach. Doing so may have benefits in terms of generating solutions of the same quality using a conceptually and computationally simpler approach._
Chapter 10
Conclusions and Future Work

*It may be right to ahead, I guess;*
*It may be right to stop, I do confess;*

The elections to the hebdomadal council, Lewis Carroll

10.1 Summary

The aim of this thesis was to present a computational method for the inductive acquisition and inference of explicit and implicit semantic design knowledge from the symbolic syntax of design formulations using an unsupervised pattern recognition and extraction approach. This method was then developed, tested, and evaluated for ways in which it could be employed for design problem reformulation tasks. Table 10-1 summarizes the reformulation tasks that were performed using REIFORM and the illustrative design domains and representation forms.

The method is based on viewing the problem of design reformulation using an unsupervised pattern recognition and extraction approach. Its development was inspired by algorithms used in the statistical natural language processing and digital image processing domains. A review of existing approaches established three design methodology oriented behavior criteria for the method. A review of findings in cognitive neuroscience and cognitive science established three design theory oriented behavior criteria for the method. These six behavioral criteria directly map onto the claims presented in Chapter 1. The performance of the method was assessed against these six behavior criteria in Chapter 9.
<table>
<thead>
<tr>
<th>Design decomposition</th>
<th>General analytical formulation, Model-based decomposition example</th>
<th>Hydraulic cylinder design, analytical formulation</th>
<th>FDT form ACS example</th>
<th>DSM form, Automotive example</th>
<th>DSM form, Aeroengine example</th>
<th>DSM form, Small and large apartment design problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification of linking variables and constraints</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modularity and integrative systems analysis</td>
<td></td>
<td></td>
<td></td>
<td>•</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aiding design “case” identification</td>
<td></td>
<td>•</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Topology design</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>Space Layout Planning</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>•</td>
</tr>
</tbody>
</table>

Table 10-1: Summary of design domains and problem reformulation tasks

10.2 Review of claims

10.2.1 Knowledge lean

Chapter 1 claimed that REIFORM is a knowledge-lean method, in the sense that it needs no other knowledge engineering than converting a design problem representation into an occurrence matrix form, and performing the steps II, III and IV on this matrix representation. Chapter 2 established this as a behavior criterion for the method and Chapter 9 assessed this claim. This claim is summarized as follows:

- The examples show that any analytical or non-analytically formulated problem can be converted into the occurrence matrix form.
- Different kinds of design formulations can be represented using the same knowledge representation format. Variables, parameters, functions, design components, or any design concept (such as space definitions in architectural design) can be represented using the occurrence matrix.
- Different kinds of relationships between these design concepts can be represented using the same knowledge representation format. The occurrence matrix based on an analytical formulation or FDT represents how variables and parameters occur in functions. The occurrence matrix based on DSM forms show how design components or sub-systems share relationships with other design components or sub-systems.
- The approach can be applied onto any of these problem formulations once the occurrence matrix form is defined.
REIFORM behaves by uncovering associative patterns between a set of symbols and expressions. Thus, any symbol or symbolic expression can be represented using the common representational form of the occurrence matrix. More generally, REIFORM uses a two-mode or dyadic representation, as well as a one-mode representation. That is, relationships between two types of elements can be captured using the occurrence matrix as well as relationships between similar types of elements. Therefore, the general implication is that any identifiable, symbolic or abstracted structural or behavioral relationship in the design domain can be used to construct an occurrence matrix. Once an occurrence matrix is in place, the rest of the method can be applied without change.

10.2.2 Training lean

Chapter 1 claimed that REIFORM is a training lean method in the sense that it can use very few or a large number of training “episodes” to acquire and infer design knowledge from design representation. Chapter 2 established training leanness as a behavior criterion for the method. Further, it also claimed that the acquired knowledge could be used in the same design experience to reformulate the same problem. Chapter 3 established behavior criterion C6 as REIFORM being able to use episodic experiences to acquire and infer knowledge. Chapter 9 assessed the method against these criteria. This claim is summarized as follows:

- The model based decomposition problem and the hydraulic cylinder problem were small problems, and demonstrated that REIFORM can be used to acquire design knowledge for any constrained general design problem or optimal design problem.
- The automotive, aeroengine, ACS and space layout examples were larger problems, and demonstrated that REIFORM scales up as the complexity of the problem increases, both in terms of coupling between design concepts as well as the size of the problem in terms of the number of design concepts.
- All problems demonstrated that the knowledge acquired and interpreted from a single problem representation could be used to reformulate the same problem.

The training lean characteristic derives from the nature of SVD and dimensionality reduction steps of REIFORM. SVD is known to behave well on sparse matrices. Thus, even when the “training” data is sparse, the method is able to re-represent all the design concepts, whether in event or episode form, to be represented in the same representation space (the US and SVT spaces), and is able to use this representation to uncover the implied relations. That is, all the knowledge concepts forming a part of the design experience can be projected into a common representation space, and their relationships measured. This implies that it can produce generalized abstracted knowledge over many events and episodes. Further, if events and episodes are added or taken away, then the state of this abstracted global knowledge changes...
to reflect that the event – episode relationships have changed, and therefore the global meaning abstraction must change.

10.2.3 Generality

10.2.3.1 Positive results

Chapter 1 claimed that REIFORM is a general method, in the sense that it can be applied onto problems from any design domain, any standard representational form, and varying problem complexity. Further it claimed that REIFORM can be used for performing a variety of reformulation tasks using the same method structure. Chapter 2 established generality as a behavior criterion for the method. This claim is summarized as follows:

- REIFORM’s application on analytical and non-analytical formulations was demonstrated.
- REIFORM’s application on problems from different design domains was demonstrated.
- REIFORM’s application on problem’s of varying complexity was demonstrated.
- REIFORM’s application on different types of design semantic variables (variables, parameters, functions, components, etc.) was demonstrated.

Different types of reformulation tasks were demonstrated using these illustrative examples. In each case, a comparison was made to the solution provided by the source paper from where the problems were taken. It was shown that REIFORM provided solutions of equivalent and comparable quality, and, for some cases, in a conceptually simpler manner. The reformulation tasks that were presented shared a general underlying commonality – they were all tasks that could be performed by assessing the interaction or coupling or association strength between variables and functions existing in the syntax of the problem formulation, and converting this into a measure of conceptual “similarity” between variables and functions. This enabled the same method to be used for different reformulation tasks. Design decomposition analysis clusters variables, functions or system components in terms of their coupling strength. Modularity and integration analysis assesses the modularity of a sub-system or component based on the coupling strength. Topology modeling measures the desirability or undesirability of connectivity or adjacency between two design components (for example, architectural spaces) in terms of the coupling strength.

A special use for the method lies in the identification of potentially incorrect or missing qualitative design relations within a design model in the non-analytical case. A large scale system level DSM may be the result of qualitative modeling (for example, collaborative interaction or dependency modeling performed by a team of engineers and
designers who are experts in the sub-system domains and aim to model system dependencies as a whole). In such a case, the method can be used to identify any potential incorrect or missing qualitative design relations that may or may not have been introduced at the modeling stage. As demonstrated, the method identifies clusters by blocking the similar variables and design components together. These blocks or clusters derive from explicit DSM relations that are used to induce implicit ones – together the explicit and induced implicit relations reveal the clusters. All elements that comprise a sub-system show up as a block. If there is a block identified where a large number of elements show high correlations with each other, but a sparse number of entries show low correlations between some elements, then this reveals that in the original DSM model, these elements are not positively related, or negatively related, implicitly or explicitly. The design implication is that two elements within a sub-system are poorly related. If this is by intention, no more needs to be done. However, this also may be the result of incorrect or missing relational data. In the latter case, the designers can further investigate the interaction and dependency relations between the elements showing low correlation within an identified block. This may lead to a reformulation of the original DSM.

10.2.3.2 Limitations

Some limitations were also identified. REIFORM’s flexibility in being used for various types of problem modeling and reformulation tasks also presents an issue in terms of the human intervention required in the method – human designers need to interpret the results and use them for the objective that they define. In a way, this allows REIFORM to be used as an exploratory tool for problem formulation and reformulation. However, in another sense, it implies that the results that REIFORM returns have to be interpreted to be useful to human designers. For this reason, even though REIFORM provided solutions equivalent and comparable in quality to the solutions provided in the original referenced problems, there is no mathematical proof presented in the thesis that these are “optimal” solutions. This is an area of future work, and Section 10.3 presents some ideas on this.

10.2.4 Capturing multiplicity and invariance of design knowledge

10.2.4.1 Positive results

Chapter 1 claimed that REIFORM can be used to infer multiple reformulation decisions from the same problem representation. It also claimed that REIFORM has the potential to preserve the invariant aspects of design knowledge in a design representation. Chapter 3 established three behavior criteria for the method that were necessary to measure this: C4 focused on dynamic associative relationships between symbols; C5 focused on explicit and implicit
meaning captured by symbols; and, C6 focused on knowledge acquisition over episodic experiences. This claim is summarized as follows:

- Varying the cosine threshold and the number of clusters in the K-means algorithm allows REIFORM to extract multiple patterns from the same representation that lead to valid, well-formed reformulations if they exist in this space. For example, in the house design example, different topologies emerge between spaces if different cosine thresholds are considered.

- Varying the number of dimensions retained ($k$ value) in the dimensionality reduction step allows REIFORM to capture multiple implicit relationships in the approximated $k$ reduced spaces that lead to valid, well-formed reformulations if they exist in this space. All the examples demonstrate this aspect.

- The explicit invariant knowledge that is representative of strong design relationships persists regardless of cosine threshold and $k$ values. For example, the modularity and integration decision in the aeroengine problem persisted through all $k$ levels considered, even though the patterns of modularity or integration strengths between the sub-systems changed.

The cosine threshold, number of clusters in the K-means algorithm, and the $k$ value in the dimensionality reduction step were left as user controlled parameters to retain REIFORM’s role as a method to explore alternative formulations. This provides a designer with the flexibility to observe multiple well-formed reformulation patterns. Heuristics were provided for choosing “good” values for these parameters.

### 10.2.4.2 Limitations

There is no exact “rule” provided that can ensure the selection of an “optimal” $k$ value or cosine threshold, i.e. the $k$ value that provides the best reformulation, e.g. the optimal decomposition. Again, human intervention is required in order to interpret the results – explore $k$ in the range of 2 to $a$ (Chapter 8) values till a well-partitioned matrix or a well-clustered set is revealed. This is a strength as far as exploratory modeling is concerned (as in the topology design case), but a limitation if a designer wishes to use the tool as an “exact” or “optimal” reformulation method. In its current form, there is no formal theoretical proof that the solutions it produces are the optimal ones, though there is empirical proof, as demonstrated over a number of illustrative examples, that each result produced matched with the “optimal” results quoted in the source papers. This is left as future work and some ways in which this can be done are discussed.
10.3 Future work

A number of interesting future possibilities were identified in the thesis. These are all presented in the order of the most specific to the most general.

10.3.1 Method extensions

In its current form, REIFORM has four parts – data representation, SVD analysis, dimensionality reduction and unsupervised clustering. Chapters 5, 6 and 7 show that the number of retained dimensions $k$ to produce an approximation, and the cosine threshold or the number of clusters in K-means algorithm are user controlled parameters. It is these two steps that allow REIFORM to be used as an exploratory tool for problem reformulation tasks, and the primary aim of the thesis was to develop a method that could inductively infer semantic meaning from syntactical representation. However, one might ask, how the capability of REIFORM could be enhanced in terms of ensuring that it shows a “correct” or “optimal” problem reformulation result, and so extend its use as a more exact problem reformulation method. This section presents some ideas on how this can be made possible through some extensions and additions to the method structure of REIFORM.

10.3.1.1 Developing an optimal decomposition method

Research into a broad class of graph partitioning and spectral methods ((Michelena & Papalambros, 1997)) reveals that using a different interpretation of the eigenvalue-eigenvector structure of a graph adjacency or Laplacian matrix can be used to infer optimal decomposition results. Though the role of SVD seems to be unexplored in this context, there is a theoretical connection between the SVD decomposition and the Eigenvalue Decomposition (EVD) of matrices, with SVD being the more general case for rectangular matrices. If the design decomposition problem were posed from a spectral methods perspective, then the role of SVD in developing a new decomposition method could be an immediate future extension. The connection between the mathematical properties deriving from SVD proofs and design decomposition tasks posed in general rectangular matrix representation needs to be explored further – the work presented in the thesis shows that it may be possible to engage in a full theoretical study that provides proof that the decomposition provided by SVD is an optimal one.

10.3.1.2 Applying soft K-means algorithm for identifying overlapping clusters

The cosine threshold and the number of clusters in the K-means algorithm are parameters that use the degree or intensity of cosine coupling strength to produce clusters that show the problem decomposition. In the clusters that they produce, each variable or function is allowed to be part of only one cluster. However, in complex design problems, the degree of coupling
could be such that often it would not be possible to identify that a certain variable or function is part of just one cluster. It may prove to be beneficial to develop a method that can assess the “belongingness” of a certain variable or function to a cluster, with the idea that a variable or function could belong to several clusters simultaneously, with a certain quantitative measurement of “belongingness” to each cluster.

The decomposition that is chosen directly affects the cost of solving the problem in terms of solving the independent sub-problems as well co-coordinating the solutions to solve the full master problem. For example, if the system is highly coupled, it may not be possible to solve all the sub-problems independently and solutions produced by one sub-problem could be an input into others and vice versa. In such a case, it will be useful to have some insight into the degree to which each variable or function is coupled to the sub-problems being considered, i.e. a measure of how much one sub-problem affects the other. To assist with providing insights more flexible decomposition possibilities for highly coupled systems, a future extension of REIFORM could be to use another version of the $K$-means algorithm called Soft $K$-means clustering (Mackay, 2003). Soft $K$-means algorithm allows a data point to be part of multiple clusters with different degrees of “belongingness”. Depending on the number of clusters (or cosine threshold) being considered, different overlapping clusters are produced, since one variable or function is allowed to be part of several clusters with varying intensity.

10.3.1.3 Developing a probabilistic approach in REIFORM for optimal “$k$” (dimensionality) assessment

Chapter 2 described how the development of REIFORM was analogically based on the use of SVD and dimensionality reduction algorithms used in the natural language processing method LSA. One limitation of the language based LSA approach reported in the literature is that, in this approach, an “optimal” value of $k$ cannot be ascertained, and it has to be left as a parameter. The thesis has shown that, for design problem reformulation, this is not necessarily a limitation because analyzing a design problem over a range of $k$ values (refer to the discussion in Chapter 8, $k = 2$ to $a$) could provide multiple “correct” reformulations. For example, in the aeroengine problem, an analysis of how coupling strength changes over a varying range of $k$ values could identify the High Pressure Compressor (HPC) system’s identity in terms of a degree of modularity rather than as simply modular or integrative. As another example, in the house layout problem, different connectivity patterns or topologies could be derived using varying cosine thresholds and $k$ levels.

However, there could be cases in which an “optimal” $k$ value could be useful – especially if REIFORM is to be used as an exact reformulation method guaranteed to find an optimal solution, instead of a heuristic method that provides a range of “good” solutions. A second
approach to consider (in addition to the one presented in Section 10.3.1.1) would be the
Probabilistic Latent Semantic Approach (PLSA) (Hofmann, 1999) that approaches the same
problem from a statistical foundation. In the PLSA approach, occurrence data from any
dyadic domain (but most noticeably from the language domain, i.e. words and documents) is
cast in terms of probability distributions over words and documents. A latent class variable is
defined that relates the probabilities of occurrences of words and documents. The number of
latent class variables is less than either the number of words or the number of documents, and
therefore, the interpretation for the latent class is something like a “topic” or “subject” to
which a certain set of words or documents relate to. Since the number of latent class variables
is less than the number of words or documents, this performs a dimensionality reduction.
Because of the statistical foundation, and other comparisons to LSA that can be found in
(Hofmann, 1999), a main strength of the PLSA approach is that it can find the optimal value
for the dimensionality reduction operation.

Developing this for design problems, notice that the occurrence matrix used in this thesis
is also a general dyadic domain. The interpretation for developing a PLSA-like approach for
the variables and functions is that the sub-problem clusters are like the latent class variables.
That is, the process predicts with what probability a variable or function is part of a sub-
problem. Since the dimensionality reduction step in this process is assured to be optimal, the
decomposition would be an optimal one.

An attempt was made to develop a PLSA like approach in this thesis, but was not able to
proceed due to paucity of data to derive probability distribution and the cost associated with
creating synthetic problems to deduce the distribution. The probabilistic approach depends on
the availability of a large body of data to develop the latent class model, and, while it is easier
in the language domain to find enough data to develop the model and then test it, this was
much more difficult for design problems. Design problems have the further constraint that the
symbols are, in a statistical sense, domain defined – so the data that is used to develop such a
model would have to be from the same domain.

10.3.2  Method application to new areas

10.3.2.1  Unsupervised learning over a design database: providing formulation assistance

REIFORM in its current form can perform episodic “learning” or inference over many
functional relationships in a single design formulation. Developing the model one step further
and using the same method structure, there is nothing to stop it from being used at the next
level – for performing unsupervised learning over a design database of multiple design
formulation examples.
Consider the following formulation. Instead of a variable by function matrix, consider an occurrence matrix in which the rows are the variables and functions and the columns represent the design formulations. That is, each matrix entry is a measure of whether or not a variable or function appeared in a design problem representation. Assume that the training database (this occurrence matrix) is an incremental one. That is, as a new hydraulic cylinder problem is solved (say a multi-objective formulation as in (Michelena & Agogino, 1988)), its final formulation becomes part of this matrix as a new appended column. Also assume that this occurrence matrix contains all the problems from all the domains that the system has seen till date. Performing REIFORM on this large occurrence matrix would then produce a representation space in which all hydraulic cylinders will be clustered together in one part of the space, all aeroengine problems in another etc., because similarity between two problems is represented in terms of distance in this method. That is, similar problems would cluster together in space. Note that within the same industry, the problems could be closer to each other in type and share variables and functions and will not be as diverse or different as the illustrative examples presented in this thesis. One precaution in formulating the occurrence matrix would be to preserve the identity of the design concepts while representing them. That is, two variables representing different concepts should not be named with the same symbols.

For a new problem that is being reformulated, a designer could query this global space of problem representations. REIFORM could, in principle, extract all the relevant variables, parameters and functions for that domain or type of problem, giving the designer an insight into what kinds of groupings of variables and functions were considered in formulating or reformulating other problems of a similar type. The answers that REIFORM would provide would be a generalized abstraction over all the problems that it has seen to date because the method structure of REIFORM causes it to represent the problems in one global space. In terms of a distributed memory system, all problems have measurable relationships with each other, even though they occurred as separate episodes. Thus, the answers returned by REIFORM would be abstracted over many “experiences” represented in the same general space.

This extension of REIFORM could be used to provide formulation and reformulation assistance in terms of selection of related design variables, parameters and constraints in design problems of a similar type. This is one example of how REIFORM could be used for performing knowledge transfer within the same knowledge domain, between similar design problems. Similar to the PLSA idea, an attempt was made to model this extension. However, for similar reasons, to test the plausibility of the idea, large amounts of credible data was required, i.e. design problem formulation data for many examples of similar types from similar design domains. This was difficult to obtain in the time period allotted, and hence this extension idea could not be tested. A very small preliminary version was tested with different
versions of the available hydraulic and explosive cylinder examples, and the results seemed promising.

10.3.2.2 Analogical or cross-domain knowledge transfer

Using a general perspective, occurrence matrices capture the relationships that two types of concepts have with each other in a certain knowledge domain. Therefore, if two occurrence matrices from different domains are similar, or begin to look similar in the $k$-reduced approximation space in terms of relationships between concepts, then this could provide an opportunity for cross domain knowledge transfer. To ground this idea in terms of a practical example, consider the following examples: (a) a set of mathematical equations representing simple harmonic motion and damping; (b) a set of equations describing a mechanical spring system; (c) a set of equations describing an AC electrical circuit. The basic mathematical formulations of these examples are the same, because the latter two would be based on the first one in terms of physics principles and behavior. It is reasonable to assume that the occurrence matrices, or the $k$-reduced approximations would be similar in these three cases. Therefore, an analogical mapping between design concepts and relationships in these domains and the application of REIFORM could allow concepts and relationships from one domain to be used in another. If concept or relationship $A$ from one domain is similar to $B$ in another domain, and if $C$ is retrieved as semantically related to $B$, then $C$ or the analogical counterpart of $C$ could also be considered semantically related to $A$. This extension is an idea on how REIFORM could be extended to perform cross-domain analogical knowledge transfer.

10.3.2.3 An interactive system for symbolic reformulation and numerical optimization

In its current form, REIFORM has been shown to perform symbolic problem reformulation. If this became a part of a larger system which had a formulation and reformulation unit interacting with a numerical optimizer, then reformulation using REIFORM could be modeled as a complete interactive process. The designer could use REIFORM to reformulate a problem symbolically, and then use a numerical optimization algorithm over this model. If the results are not satisfactory, this information could be fed back into REIFORM, and a changed occurrence matrix or different parametric settings could be used to reapply REIFORM and so on. In combination with a more powerful numerical optimization system, in which the designer has access to a range of optimization algorithms, this interactive process would prove to be much more useful. In problem decomposition tasks for example, it is usual for different decompositions to require different sub-problem-main-problem solution coordination strategies which have a direct implication on the cost implied by the decomposition in solving the problem, as well as different sub-problems requiring different solution
algorithms. In such cases, the combination of REIFORM with the numerical optimization system would aid the decision making further.

10.3.3 Design theory: “Global” design computation methods and the role of statistical, unsupervised methods

The research literature review shows that a great number of design computation methods are based on supervised learning methods or AI methods that depend on direct, local techniques of symbolic manipulation. In the domain of pattern based approaches, evolutionary computing methods such as genetic algorithms offer one avenue of performing pattern-based explorations, guided by a global objective function.

SVD, dimensionality reduction and unsupervised clustering are just a set of tools used from a large bag of statistical analysis tools that could be used for a global pattern based analysis in design computation. Machine learning and AI algorithms based on supervised learning and knowledge-rich approaches have been explored in much depth in design computation literature. A general future direction that REIFORM reinforces for design theory is the development of design computation and automation methods that derive from statistically based unsupervised machine learning techniques.

The work in this thesis shows that it is possible to use unsupervised pattern recognition and extraction based approaches to perform inductive inference of design knowledge from design formulation data. By simply changing the “theoretical” standpoint on how a “symbol” is defined in design representations, it was possible to use a simple set of mathematical ideas to perform such unsupervised inductive inference for a knowledge domain that traditionally relies on “giving” the system complex design domain and task knowledge for performing the same set of tasks. In summary, one general lesson provided by the performance of REIFORM is that the way in which a set of symbols encodes and generates semantic meaning may be a higher order phenomenon than a one-to-one direct mapping between symbol and meaning. Therefore, an extension of the symbolic “rule based” AI perspective could incorporate another lower level “rule” – the distributed associations that exist between symbols and their co-occurrence patterns analyzed from a statistical pattern extraction perspective can reveal the invariant and variant semantic meanings in design representations.

From a design theory perspective, this would also address theoretical issues concerning the computational modeling of cognitive processes and properties that designers employ, in effect, semantic syntactic transformations, many of which are not well understood.
10.4 In the end

The source of fun and pleasure in this piece of research lay in the challenge of being able to use a set of particularly simple mathematical ideas to model a somewhat difficult design theory question with no well-known cognitive or computational models – the transformational connections between syntax, semantics and symbol processing as they occur during symbolic problem modeling and reformulation tasks in design. The mathematics is by no means “complete”, as are not the answers to the difficult questions. But a certain joy derives from the fact that there is much more to be done, and that, maybe, a small part of it was done in these pages.


References


Appendix A: List of Matlab programs

All file names are self-explanatory and contain the relevant Chapter number and the example description.

Chapter4and5DecompositionExample.m
Chapter5ACSExample.m
Chapter5ExampleDSMAutomotive.m
Chapter5ModularSystemsExamplePrattWhitneyModel.m
Chapter6LargeApartmentTopologyExample.m
Chapter6SmallHouseTopologyExample.m

cosineMeasure.m
Function that computes cosines between two vectors.

kReduced.m
Function that computes a dimensionality reduction over the SVD of a matrix by preserving the first $k$ singular values.

MatrixReorderXY.m
Function that computes the row and column ordering on $X(k)$ and $Y(k)$ for the matrix partitioning algorithm. The row and column ordering are then used in MatrixReorderZ.m.

MatrixReorderZ.m
Function that computes the final row and column ordering on $Z(k)$. Uses the output row and column ordering from MatrixReorderXY.m.