Learning Symbolic Formulations in Design: Syntax, Semantics, Knowledge Reification

Somwrita Sarkar, Andy Dong

University of Sydney, Australia

and

John S. Gero

George Mason University, USA

Corresponding Author:

Somwrita Sarkar

Design Lab, Wilkinson Building,

Faculty of Architecture, Design and Planning,

University of Sydney, NSW 2008

Email: ssar3264@mail.usyd.edu.au

No. of manuscript pages: 45

No. of tables: 0

No. of figures: 16

Learning Symbolic Formulations in Design: Syntax, Semantics, Knowledge Reification

Abstract. An AI algorithm to automate symbolic design reformulation is an enduring challenge in design automation. Existing research shows that design tools either require high levels of knowledge engineering or large databases of training cases. To address these limitations, we present a singular value decomposition (SVD) and unsupervised clustering based method that performs design reformulation by acquiring semantic knowledge from the syntax of design representations. The development of the method was analogically inspired by applications of SVD in statistical natural language processing and digital image processing. We demonstrate our method on an analytically formulated hydraulic cylinder design problem and an aero-engine design problem formulated using a non-analytic Design Structure Matrix form. Our results show that the method automates various design reformulation tasks on problems of varying sizes from different design domains, stated in analytic and non-analytic representational forms. The behavior of the method presents observations that cannot be explained by pure symbolic AI approaches, including uncovering patterns of implicit knowledge that are not readily encoded as logical rules, and automating tasks that require the associative transformation of sets of inputs to experiences. As an explanation, we relate the structure and performance of our algorithm with findings in cognitive neuroscience and present a set of theoretical postulates addressing an alternate perspective on how symbols may interact with each other in experiences to reify semantic knowledge in design representations.

Keywords: machine learning in design, symbolic problem reformulation, singular value decomposition, pattern extraction, unsupervised clustering

1 Introduction

Problem reformulation in design is concerned with the metamorphosis of design semantics into a formal mathematical model. Design semantics are the "meaning" of a design work as intended by a designer – structure and behavior of a design object are conceived in terms of symbolic elements and relationships. A mathematical model reifies the design semantics as symbols and mathematical functions. The development of AI algorithms to automate design problem reformulation tasks is an enduring challenge in design automation. Existing methods either require dependence upon high levels of embedded knowledge engineering in the form of rules, heuristics, grammars or domain/task specific procedures (e.g., (Campbell et al., 2003; Ellman et al., 1998; Gelsey et al., 1998; Medland & Mullineux, 2000)) or require a large database of training cases (e.g., (Duffy & Kerr, 1993), (Schwabacher et al., 1998)). It would be useful to develop a method characterized by the following desirable features: (1) a knowledge-lean method that does not need any significant design domain or task knowledge to be embedded into the system; (2) a training-lean method that can extract design knowledge over one or very few cases; and, (3) a simple and computationally efficient method applicable over different design domains, representational forms (analytical, non-analytical, etc.), problem sizes and complexity (small, medium, large, etc.).

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 (Papalambros & Wilde, 2000): which design elements to represent as variables, which ones to fix as parameters, what relationships between variables and parameters to consider as objective functions or as constraints, how to decompose a large, over- or under- constrained design problem, how to reformulate problems into mathematically simpler forms so that they become easier to solve, etc. Once a mathematical representation is constructed, numeric algorithms are applied to a defined problem space, and the optimal or feasible solutions found (even though reformulation and solution search are cyclic, iterative processes).

Yet, how are these representations reified in the first place or reformulated in subsequent phases of designing? It is known that designers encode the "meaning" of a design object in the design representation through symbols. They construct such representations on the basis of experience-based knowledge and current design problem requirements. However, from an AI standpoint, there is no obvious rule-based answer to their approach. There is no known computational process that takes in abstract modeling requirements as input and produces a mathematical representation as output. This is because a mapping between design representation syntax and the encoded design semantics is not a simple one-to-one direct one but a complex and multi-faceted one. The authors believe that trying to go from abstract semantics to syntax is a too hard and general problem to tackle by known machine learning and AI methods. 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*.

2 An analogical inspiration for the proposed method

The inspiration for the method came from an analogy we identified between the Latent Semantic Analysis approach (LSA) (Landauer & Dumais, 1997) from statistical natural language processing (SNLP) and image compression methods (Kalman, 1996;Strang, 2003) from digital image processing (DIP). Both involve use of the linear algebra based matrix factorization method of Singular Value Decomposition (SVD). In SNLP, SVD-based LSA is used to reveal semantic patterns in textual data by analyzing contextual occurrences of words in sentences 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 an intriguing connection between semantic knowledge and its syntactic representation. As stated in Section 1, *a symbolic-mathematical design representation is the description of semantic design knowledge. Structural elements in behavioral relationships are encoded as symbols in functions in a syntactic representation.* Using the mappings from SNLP and DIP to design problem reformulation, we

conjectured that SVD would be able to reveal the connections between the syntax of design representation and intended design semantics. We use structural and behavioral analogies to explain the mappings to design.

2.1 Structural analogies

2.1.1 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 the words appear (Figure 1(a)). The matrix entries are a measure of the number of times a word appears in a specific document. If we draw 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 elements (variables, parameters) and columns would represent relationships between these elements (objective functions, constraints). The matrix could thus represent the semantic "meaning" of the design through relationships between variables and functions. Figure 1(b) shows an example formulation – an analytic, non-linear, single objective optimization model for a hydraulic cylinder. Figure 1(c) shows the matrix representation for this problem. Each entry measures whether or not the variable/ parameter occurs in a function. We call this the occurrence matrix **A**. The semantic meaning is recast through this occurrence matrix by capturing the associative patterns between sets of elements (words or variables, and, in general, components of a domain) to experiences (documents or functions, and, in general, syntactic structures comprised of those elements).

In LSA terminology (Landauer & Dumais, 1997), the matrix captures how *events* occur in *episodes*. 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" the relations. Michelena and Papalambros (1997), for example, present a review of these methods, and present a hypergraph based algorithm that decomposes a design problem into weakly connected sub-problems.

The occurrence matrix form can be used to represent both analytically and non-analytically formulated problems. The Functional Dependence Table (FDT) form (Li & Li, 2005) is an established method of representing an analytic formulation as a non-analytic matrix formulation, or of representing relationships between design variables based on results from numerical simulations. It has a direct structural correspondence with the word-by-document matrix in LSA. In this general structural analogy, the matrix measures how elements occur in the local context of one another. 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.

Figure 1: (a), (b), (c)

2.1.2 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 size $m \times n$; the mn matrix entries contain measurements of pixel values (Kalman, 1996), Figure 2(a)). 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) ((Sosa et al., 2003), Figure 2(b)). The example DSM representation shows an aero-engine problem with 54 design components in 8 subsystems. Similar DSM representations also exist for mappings such as tasks and processes. 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, however, the matrices correspond to relationships that exist between the same elements within a knowledge domain in rows as well as columns. While image matrices can be square or rectangular, DSM matrices are always square.

Figure 2: (a), (b)

2.2 Behavioral analogy

However diverse the structural analogies may appear, they seemed deserving of a deeper analysis. We found that the same approach is also a widely used one in many other knowledge domains such as design text, content and team performance analysis (Dong, 2005), prediction of psychological phenomenon (Wolfe & Goldman, 2003), internet search algorithms (Strang, 2003), and clustering gene microarray data (Liu et al., 2003). We, therefore, turned to a behavioral analysis of the mathematical structure of SVD to draw a behavioral analogy.

SVD factorizes a general rectangular matrix **A** with m rows and n columns by decomposing it into a product of three matrices, $\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^{\mathrm{T}}$, where **U** ($m \times m$) and **V** ($n \times n$) are the left and right singular matrices. **S** ($m \times n$) is a rectangular matrix with r non-negative singular values that capture the dominant association patterns in the data in decreasing order of magnitude. The number of singular values is r, where r is the rank of **A**. The mathematical idea ((Strang, 1993, 2003), Figure 3) is as follows: the row space of **A** is r-dimensional and inside $\boldsymbol{R}^n$, and the column space of **A** is r-dimensional and inside $\boldsymbol{R}^m$. We choose special orthonormal bases $\mathbf{V} = (\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots \boldsymbol{v}_r)$ for the row space, and $\mathbf{U} = (\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots \boldsymbol{u}_r)$ for the column space, such that $\mathbf{A}\boldsymbol{v}_i$ is in the direction of $\boldsymbol{u}_i$. s_i provides the scaling factor and $\mathbf{A}\boldsymbol{v}_i = s_i\boldsymbol{u}_i$. In matrix form, this becomes $\mathbf{AV} = \mathbf{US}$ or $\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^{\mathrm{T}}$. Thus, a general rectangular matrix **A** is diagonalized into two independent spaces represented by special orthonormal bases **U** and **V**, and related to each other by the magnitudes of the singular values. A well known result in linear algebra is that if a dimensionality reduction is performed on this decomposition using the first k largest singular values, then this produces a linear least squares

approximation of **A**. This dimensionality reduction step strengthens the most important association patterns of matrix **A** (through the first k singular values) and weakens the less important ones, treating them as noise (the $r - k$ singular values).

Figure 3: (a), (b)

2.2.1 What does SVD do in LSA?

A word-by-document matrix **A** contains measurements of local occurrences of words in documents. SVD of this matrix **A** produces factors **U** and **V**. These are new, abstract, orthogonal bases with orthonormal components derived from mutual co-occurrence information contained in the original data matrix. **U** and **V** represent abstract "word" and "document" spaces. The vectors in **U** and **V** measure correlations between the elements in the original co-occurrence matrix, but are themselves uncorrelated to each other. In other words, **A** has been diagonalized into a word-space **U** and a document-space **V**. The singular values in **S** capture the dominant patterns of association in matrix **A** in a decreasing order of magnitude. All the words and documents can now be represented as linear combinations of these orthonormal vectors. In a dimensionality reduction step, a k-reduced linear least squares approximation of the original matrix is produced. The original data is now observed in a reduced number of dimensions.

The conceptual explanation for this mathematical process is based on the notion that linguistic knowledge contains a large number of weak interrelations. Words that appear together in sentences or documents share semantic similarity – they occur together in sentences to capture meaning. The dimensionality reduction step induces implicit relationships ('latent' in the LSA terminology) based on explicit local relationships between words and documents. The matrix **A** shows local explicit relationships between words and documents. However, implicit global relationships can be induced through the dimensionally reduced matrix $\mathbf{A}'$ as the result of a computation that "globally" takes into account all the associations of all the words with all the documents. Words and sentences expressed as linear combinations of orthonormal vectors can be plotted as vectors in real space. Semantic similarity is assessed between them through cosine measurements. This reveals

the semantic relationship of a word with a document, even though the word may not occur directly in the document. A principal claim in LSA is that choosing a "correct" dimensionality (lower than the original) makes it possible to observe these implied patterns in the data that are not directly observed in the original data. For example, this computation will capture the semantic relationship that "SVD" shares with "linear algebra", even though it may not appear directly in the title of a book "Introduction to Linear Algebra". If "SVD" is used as a search term instead of "linear" or "algebra" in searching for a book on linear algebra, the query should still bring up books on linear algebra. In LSA, it is the degree of approximation that is interesting. There is no attempt to reduce the error between the approximation and the original matrix because the claim is that the approximation reveals patterns that cannot be observed in the original data. A larger error might actually be useful to uncover latent relations.

2.2.2 What does SVD do in image compression?

In image compression, the focus is on producing a "least lossy" approximation of the original matrix **A** because the objective is data compression. The matrix **A** ($m \times n$) is a representation of pixel values. 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. This has a direct behavioral analogy with images. Any large scale feature in the image will tend to show redundancy, as rows and columns will contain similar repeated values to represent this feature. This implies that an 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 high degree of approximation. Note the difference in interpretation from LSA – in image compression, error reduction and a "least lossy" approximation is the aim.

The mathematics is relatively simple. SVD is performed on matrix **A**. Then, an optimal reduced rank approximation is found that is "good" enough to represent the original data. A common "rule of thumb" measure is that the human eye should not be able to make out the loss in data.

2.3 Behavioral characteristics from SVD and dimensionality reduction

Based on the above discussion, the following behavioral characteristics are interesting for extracting semantic meaning from design representations:

2.3.1 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 possibly left implicit, i.e., they are not explicitly represented. Consider a simple example – the concepts of *area* (*a*)*, volume* (*v*)*, length* (*l*)*, breadth* (*b*) *and height* (*h*). In a representation, two functions could be $a=l*b$ and $v=l*b*h$. However, that *volume* is also *area* times *height* is an implied, latent or weak relationship that exists in the semantic space but is not explicitly stated in the representation. In fact, it is often not explicitly stated as it could produce redundancies or over-constrain the problem, both of which are undesirable for numerical solution packages. While this is a trivial case, in general, it should be asserted that each functional representation is one possible ordering or capture of a behavior through the symbols which occur in the function. Other behaviors may weakly appear that are not included in the explicit representation. The process of projection of symbols (**U** space) onto functions (**V** space), and vice versa, done by SVD locates these weaker, latent relations. If SVD is able to extract implicit patterns in the natural language domain, then it should be able to do so for design representations. This property may be useful to recreate the semantic meaning of the design work intended by the combination of the symbol space and the function space even where this meaning is not explicitly stated. The SVD method followed by dimensionality reduction could reveal patterns contained in the syntax that are not directly observable in the original co-occurrence matrix. Almost always, design representations are sparse. Symbolic design representations do not explicitly code all the semantic relationships. Thus, this is an important characteristic for design problem reformulation.

2.3.2 Redundancy, matrix compression and explicit pattern extraction

The image compression 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. For a large design problem, if the matrix contains redundancy, then this is an indication that the explicit relationships in the problem representation can be inferred at a lower approximation than the original, i.e. at some lower $k = r$ (rank of matrix **A**). In a design problem reformulation task, we need the method to find the implicit as well as the explicit relationships.

2.3.3 Inference of multiple and invariant relationships from a single representation

The above discussions show that SVD and dimensionality reduction might be able to capture both explicit-invariant and implicit-multiple 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 because they 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. We will show that there are two parameters by which designers can experiment with extraction of multiple reformulations from a single design representation – the number of singular values to retain in the dimensionality reduction step and a cosine threshold value to measure semantic similarity in the inferred design relationships. If the inferred relationships 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 re-formulation. In some design problems, there are multiple potential formulations that can be considered to be "good" formulations and no dominant formulation. By varying these parameters, different reformulations can be observed.

3 Method

3.1 Method summary

The method for problem reformulation performs inductive and unsupervised inference of semantic design knowledge from purely syntactical examples of design formulation data. Design formulation examples expressed in standard forms make up the training set. The method operates on each individual sample. The method acts as a "design formulation assistant" by: (1) re-structuring design formulation knowledge from existing experiences and re-constructing it for similar problems (problems from a common design domain, e.g. the design of hydraulic cylinders, or sharing a common mathematical form, e.g. an incidence matrix or a Design Structure Matrix); and, (2) assisting with one-shot reformulation decisions on a single problem formulation task. The designer can query the system for reformulation decisions of various types. The method operates incrementally such that the same query returns different answers as the number of samples increases.

We demonstrate the method on an analytically formulated hydraulic cylinder design problem and a large scale, non-analytically formulated aero-engine problem employing a Design Structure Matrix representation. In this paper, we show that the method can assist with the selection of variables, parameters and functions, the identification of design cases, design decomposition, and performing modularity and integration analysis for systems by identifying shared or linked groups of variables and functions. The results show that the method does not require design domain or task knowledge to be pre-specified, and is able to develop this knowledge over inductive experiences. It learns quickly over a very small data set, needing just one example (i.e., problem formulation statement) to show useful performance. It is computationally simple to implement and can be applied over different problem domains, for different tasks, over different problem sizes.

3.2 Analytic and non-analytic design representations

In this section, we describe the representational formalisms that we employ along with brief descriptions of the example problems that we use to demonstrate the method.

3.2.1 Analytic formulations

An analytic formulation is a design formulation in which variables and parameters are characterized by symbols, and mathematical functions describe the relationships between them. Common examples of analytic formulations are design optimization formulations (Papalambros & Wilde, 2000) – linear, quadratic, non-linear, continuous, discrete, mixed etc. Equation (1) describes an optimization problem formulated in its general canonical form:

$$\text{Min } \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{p})$$

Subject to

$$\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{p}) \leq \mathbf{0}$$

$$\boldsymbol{h}(\boldsymbol{x}, \boldsymbol{p}) = \mathbf{0}$$

$$\boldsymbol{x}, \boldsymbol{p} \in \chi \subseteq \mathbf{R}^n \qquad (1)$$

Here, $\boldsymbol{x}$ is the vector of design variables and $\boldsymbol{p}$ is the vector of design parameters. Parameters are quantities that are kept fixed for a particular design model. $\boldsymbol{g}$ and $\boldsymbol{h}$ are vectors of inequality and equality constraints, respectively, and set feasibility conditions. Both $\boldsymbol{x}$ and $\boldsymbol{p}$ belong to some subset χ of the real space $\mathbf{R}^n$, where n defines the total number of dimensions in terms of the numbers of variables and parameters. The functions $\boldsymbol{f}$ are a vector of objectives to be minimized; for a single objective case, this becomes a single function f.

An example analytical formulation is the hydraulic cylinder design problem (Figure 1(b)) that we use to demonstrate the method. There are 5 design variables and 4 design parameters. The objective is to minimize the diameter of the hydraulic cylinder subject to 4 inequality and 2 equality constraints. We apply our method onto this problem to demonstrate the following tasks:

(1) selection of design variables, parameters and constraints; (2) identification of design cases. We use an additional multi-objective formulation of the same problem to test whether the method can identify multiple as well as invariant design relationships by varying the parameters.

3.2.2 Non-analytic formulation: Aero-engine design problem

A non-analytic formulation is a design formulation in which design elements are represented as symbols and relationships between them are captured as binary mappings in a matrix: there exists/does not exist a design relationship between two design elements or functions. Non-analytic formulations are used frequently for system decomposition and related analysis tasks in large scale or complex problems. An analytic formulation can be converted into a non-analytic form matrix using a Functional Dependence Table (FDT). Examples of non-analytic formulations are matrix forms such as the FDT form (Li & Li, 2005) or a Design Structure Matrix (DSM) form (Rowles, 1999; Sosa et al., 2003). Figure 2(b) 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 (HPT), Mechanical Components, and Externals and Controls. The matrix elements show which components share interfaces and/or design dependencies with which other components. The method is applied to this problem to demonstrate the following tasks: (1) design decomposition, (2) analysis of modular versus integrative systems. This example is also used to evaluate the computational performance of method over large scale problems.

3.3 Method description

3.3.1 Data representation

We use the single objective hydraulic cylinder problem to demonstrate the method. The first step is to convert an analytical problem formulation, Figure 1(b), into an occurrence matrix **A**, Figure 1(c). The rows represent variables and parameters, and the columns represent the objective and the

constraints. Each matrix entry A_{ij} is set to 1 if a variable or parameter occurs in an objective or constraint and 0 otherwise. The matrix captures local patterns of occurrence of design elements in the "context" defined by the functions, where the coming together of the elements themselves defines the context.

3.3.2 SVD analysis: re-representation of the formulation

SVD on matrix **A** produces $\mathbf{USV}^T$, a re-representation of **A** in terms of new abstract orthogonal bases **U** and **V** with independent components derived from mutual co-occurrence information contained in the original data matrix. The columns of **U** represent an abstract "design element" space and the rows of **V** represent an abstract "design function" space. Each row of **U** (with r components) is an abstract representation of the m variables/parameters. Each column of **V** (with r components) is an abstract representation of the n functions. The singular values capture the dominant association patterns in **A**. Thus, **US** and $\mathbf{SV}^T$ describe a scaled design element and design function space respectively. Each design element or function can be expressed as a linear combination of the dimensional vectors in **U** and **V** scaled by the singular values. In this step, the discrete local explicit relationships between variables and functions are converted into continuous global implied relationships. The original relationships are de-coupled to produce abstract orthonormal vectors, linear combinations of which represent the variables and functions represented in real space. Figure 4 shows the SVD decomposition for the data matrix shown in Figure 1(c).

Figure 4

3.3.3 Dimensionality reduction

In the dimensionality reduction step, the first k singular values are preserved to produce a least squares approximation $\mathbf{A}' = \mathbf{U}(m\times k)*\mathbf{S}(k\times k)*\mathbf{V}^T(n\times k)$. Since the approximations based on retaining only the k largest singular values will overplay the most important associative patterns and underplay the rarer ones, variables and functions that mutually share high coupling relations in **A**

will be strengthened and those that do not will be weakened. The step causes semantically positively correlated variables (those that appear together in functions) to cluster together in this dimensionally reduced space, while negatively correlated variables (those that do not appear together) fall far from each other. This is how implicit relationships are induced using the explicit relationships. Figure 5 shows the k=2 reduced approximation of the matrix **A**. The truncated matrix **A′** shows that the entries change to higher or lower values, induced from the explicit binary relationships.

Figure 5

Consider in matrix **A** (Figure 1(c)) that the variable "internal diameter" *i* occurs in constraint *h2:s=ip/2t* measured by A_{17} = 1. It does not occur in constraint *g4:s–S*≤0, indicated by $\mathbf{A}_{15}$ = 0. But, from the algebraic relation in *h2*, we can see that the value of *i* affects the value of *s* (or vice versa) which will affect the behavior of the constraint *g4*. *i* implicitly affects and is affected by the behavior of *g4* even though it does not occur in it explicitly. This is an example of implicit knowledge that designers regularly employ. A pure symbolic AI based inference system, for example, would need to encode such knowledge as an explicit algebraic substitution rule. The matrix **A′** (Figure 5) shows that the original 0 between *i* and *g4* has changed to a higher value $\mathbf{A}_{15}'$ = 0.1987 (showing that there is an implicit relationship) and the original 1 between *i* and *h2* has changed to $\mathbf{A}_{17}'$ = 1.0957 (showing a stronger relationship than 1). The original 1-0 local entries have been re-represented in terms of a "global" map of syntactical associations that each element has with every other one.

3.3.4 Graphical representation of dimensionality reduction

US is the scaled design element space. $\mathbf{SV}^T$ is the scaled functions space. Now consider the dimensionality reduction operation in terms of the following matrix multiplications with k=2 (Figure 6):

Figure 6

X and **Y** give us the 2D reduced **US** and **SV** spaces, so each pair of entries ((x_{11}, x_{12}), (y_{11}, y_{12}) etc.) gives us the coordinates for each design element and function in a common 2D space. Figure 7 shows the graph for the 2D representation of the single objective hydraulic cylinder problem. Graphs for k=2 and k=3 help to visualize the "abstract semantic space" generated by the SVD and dimensionality reduction analysis, especially in terms of the visual relations between the design elements and functions projected into the same space. For higher k values, it is not possible to plot graphs, but the mechanics of the method stays the same.

Figure 7

3.3.5 Cosine similarity measurements: re-construction of design knowledge

The previous steps show that the method converts discrete relationships between variables and functions into a continuous position based representation in real space. Thus, distance is a measure of semantic relationship. Variables, parameters and functions that are positively correlated in the syntax will tend to cluster together in the dimensionally reduced space, as explained above. Thus, the reduced dimensionality representation can be queried using a convenient "distance" metric to get semantically related groupings of design elements and design functions. Each point in Figure 6 is a vector in 2D space that represents a design element or function. We use a cosine distance metric because it will capture magnitude as well as direction in space. Thus, a cosine angle measurement between any two vectors **x** and **y**, $\cos(\theta_{\mathbf{xy}}) = \mathbf{x}^T\mathbf{y} \,/\, \|\mathbf{x}\| \, \|\mathbf{y}\|$, is a measure of their semantic similarity – the higher the cosine angle value, the more the similarity, and vice versa. Continuing with the same example as in the above section, variable *i* has coordinates (-1.5809, -0.0483) and constraint *h2* has coordinates (-1.8929, 0.3804). The cosine between these is 0.9739. Confirming expectations, this shows that there is a high semantic relationship between them. Further, the cosine between *i* and *g4* comes out to be 0.7041. This is a high measurement, considering that in the original matrix $\mathbf{A}_{15}$= 0 for the relationship between *i* and *g4*. This serves as validation that the method brings out implicit relationships existing between design elements and functions.

A query in this space can measure three types of semantic similarity: (1) between design elements as cosines between **X** vectors; (2) between functions as cosines between **Y** vectors; and, (3) between design elements and functions as cosines between **X** and **Y** vectors. The next sections show how these queries assist with various reformulation tasks.

4 Problem Reformulation Tasks: Hydraulic cylinder design

The hydraulic cylinder design problem is an example of an analytically formulated problem. Using this example, we show the following design problem reformulation tasks: (1) selection of design variables and constraints; and, (2) design case identification.

4.1 Selection of design elements and relationships

One of the first steps in formulating a design problem are decisions about 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. Also, because 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 elements and relationships should be considered in conjunction. This semantic knowledge, as we have seen in the discussion above, is not always explicitly available in the original problem formulation. For example, should the hoop stress equation (*h4*:$s - S \leq 0$) be considered as a part of the model if *i* is chosen as a variable? Obviously, *i* does not occur in this equation, so how can one tell *a priori*?

A measurement of similarity between any one variable and all other variables and functions using a cosine of the angle between them in space can show how this variable is associated with any other. The concept of a *cosine threshold* is one parameter that a designer can use to experiment with various reformulations. Fixing a cosine threshold, all variables and functions that

show higher-than-threshold cosine measurements with the query variable/function are returned as semantically similar to the query. Recall that the local, explicit, discrete relationships in the matrix **A** have now been converted into a global, implicit, continuous distance based representation in real space. Semantic similarity is a continuous function of distance – the higher the cosine value (smaller angle), the higher the semantic similarity. Thus, a higher cosine threshold implies a 'tighter' definition and returns fewer coupled variables and functions, while a lower one 'relaxes' the definition and returns more. By varying cosine thresholds and observing different groups returned as answers, the designer can observe different possible formulations. If we fix *i* as the query variable, then Figure 8(a) shows the cosine measurements between *i* and all the other variables and functions in the *k*=2 space. If we choose a cosine threshold of 0.7, then {*t*, *s*, *p*, *g3*, *g4*, *h1*, *h2*} are returned as semantically related to *i*. If we increase it to 0.8, then {*p*, *s*, *h1*, *h2*} are returned. It is clear that the threshold of 0.8 returns only the variables and functions that *i* explicitly occurs with or in, but a threshold of 0.7 also includes variables and functions that share an implied relationship. This can be confirmed from Figure 8(b) that shows the variables and functions returned with the cosine threshold set to 0.7. The query on *i* returns functions *g3*: *p-P<=0* and *g4*: *s-S<=0*. *i* does not occur in either of these, but as we will show in the next section, they are important functions to consider for variable *i.*

A "good" choice of a cosine threshold will lead to "good" reformulations. As a heuristic for identifying good cosine thresholds, we have developed a matrix reordering algorithm that reorganizes a cosine measurement matrix (a matrix where row *i* and column *j* are represented by variable *i* and function *j*, *i*=1 to *m*, *j*=1 to *n*; matrix entry A_{ij} measures the cosine between variable *i* and function *j* in *k*-reduced space, refer to aero-engine problem presented later) to cluster variables and functions sharing similar cosines to reveal block matrices on the diagonal with similar cosine values. Alternatively, we have also applied the K-means clustering algorithm on the *k*-reduced space to reveal these semantically related clusters.

Figure 8: (a), (b)

4.2 Heuristic design case identification

The computations shown in the previous sections can be utilized to inform a specific problem reformulation task – design case identification. Any problem formulation model contains a set of constraints. This model could be over or under-constrained, i.e. not well-formulated. 'Design cases' are sets of active or critical constraints for a design optimization problem formulation that lead to a well-formulated model. One method for identifying these design cases is monotonicity analysis (Papalambros & Wilde, 2000), that is a problem-solving-by-reformulation method in which constraint activity information is used to reformulate the problem to a simpler form when a problem is over or under constrained. 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.

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 cases. Papalambros and Wilde (2000) identify 3 design cases – stress-bound, pressure-bound, and thickness-bound. Their results show that for design variable *i* (internal diameter) either constraints (*g3,* (*g2, h1*)) or ((*g4, h2*), (*g2, h1*)) will be active, and for variable *t* (wall thickness) either constraints ((*g4, h2*), (*g2, h1*)) or (*g1*) will be conditionally critical. Figures 9(a) and (b) shows that cosine measurements between function vectors in the 2D space show similar conclusions by purely a syntactic analysis of design formulation. High cosines in 9(a) are shown as connections in 9(b). Cosines between constraints (g_2, h_1) and (g_4, h_2) are high, with constraints g_3 and g_1 sharing high cosines with these two groups respectively.

Figure 9: (a), (b)

Note that monotonicity analysis is a mathematically rigorous, rule-based procedure, and it can identify these cases without ambiguity. Our method can only provide insights into design case "clusters", and cannot provide optimal solutions as monotonicity analysis does. For example, while it shows that there are groups of (*g2*, *h1*) and (*g4*, *h2*), 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. This method quickly turns into a very complex solution procedure for even a small-medium sized problem as the number of variables and constraints increase. Our method would be particularly suited for such large 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 related variables and functions. An added advantage of the method 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.

4.3 Performance analysis over multiple samples

We have shown that the method requires just one training sample to extract implicit design knowledge. However, a designer may wish to apply the method on several samples of a problem in a specific design domain to explore the implied knowledge returned in these separate cases. We have claimed that the method is training lean, and can identify the multiple as well the invariant aspects of design knowledge, whether it is applied onto one or several design formulation samples of a given design problem.

To analyze the performance of the method over multiple samples, we applied the method on an additional formulation of the same problem. There are multiple ways in which the same problem can be formulated by different designers. Figure 10 shows a multi-objective formulation of the hydraulic cylinder problem (Michelena & Agogino, 1988), with conflicting objectives and a slightly different set of constraints and parameters than the single objective formulation.

Figure 10

The main conclusion to report is that the method generalizes very quickly over very few training examples. By generalizing, we mean that similar and consistent answers are returned for different variations in formulation for the same problem. As seen in the previous section, the method needs just a single formulation to infer the "correct" answers. The answers returned from the multi-objective formulation reinforce this result. The method correctly identifies semantic groupings of variables and constraints within design cases. This is interesting because, in general, computational design support systems require either a very high level of knowledge engineering, or they require a large database of training examples. 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. The advantage provided by this method is that it requires only one sample to infer the explicit and implicit design knowledge. Any other samples, incrementally added to the "training database" serve to confirm that the answers are "correct". All samples are "real" design experiences that either reinforce or change the learning gained from previous samples. The same set of cosine similarity measurements on these 2 samples reveals that the method returns "correct" answers. For example, parallel to the design case identification task in the single objective case, Michelena and Agogino (1988) use monotonicity analysis to identify three design cases for the multi-objective problem: Case P (pressure inactive), Case S (stress inactive), and Case PS (pressure and stress inactive) with all of the cases having force f and thickness t active. The SVD based method applied onto this example shows (Figure 11, k = 3) that that pressure and stress groups are distinctly apart, with force and thickness falling in the middle (as supported equivalently by cosine or k-means calculations).

Figure 11

Because the algorithm is based on extraction of syntactic patterns, it does not discriminate between "wrongly formulated" and "correctly formulated" examples. However, it can be conjectured that over many numbers of design formulation samples, a statistical effect will ensure that the algorithm generalizes "correctly", assuming that most of the examples provided to the algorithm are "correct" from the design point of view.

The method can be used to capture multiple patterns from the same example by varying the cosine threshold and the *k* value. The cosine threshold parameter has been discussed in previous sections. We now present the effects of the second parameter – changing the *k* value (the number of dimensions retained in the dimensionality reduction step) to observe multiple patterns or invariance in formulations. We examine the effect of design problem size on the answers returned, and the number of dimensions *k* that are retained while performing the dimensionality reduction as compared to the number of "correct" answers the algorithm returns. The relevance or "correctness" of the answers returned by the queries was assessed based on how the reformulation suggested by the answers matches up with those provided by the documented results reported in the source paper. For example, in the previous section we present a comparison of the answers returned by the queries from our method with the well-known and documented monotonicity analysis applied to the same problem. Since monotonicity analysis is mathematically proven and guaranteed to find the optimal solution in this case, and our results match those provided by the monotonicity analysis, it may be safely claimed that the answers returned by this algorithm are correct. By studying the effect of problem size and the value of *k,* we also assess the multiple patterns that are returned as answers, and the invariance of design knowledge across these answers.

4.3.1 Effect of problem size

Keeping the number of dimensions *k* fixed, the method pulls more variables, parameters, objectives and functions as the size of the problem increases. For example, the multi-objective version of the problem contains more variables and constraints than the single-objective version. If

we keep the number of *k* dimensions fixed for both design formulation samples, the number of answers returned to the query variables in the multi-objective case is higher than in the single objective case. Figure 12 shows this result, as over two examples of the same problem with different formulations, we see that the method manages to capture the variances in the formulations. The single objective formulation has an original matrix size of 9 × 7 (Figure 1(c)). The multi-objective formulation of the same problem has an original matrix size of 17 × 10, as the multi-objective problem has an increased number of variables, parameters and objective functions. The method returns a higher number of semantically related groups of variables and functions as answers in the multi-objective case.

This is an interesting result – it provides proof that the method scales up well. As problem size and the complexity of interaction between variables and functions increases, so will the number of implied relationships. If the method scales up well, it should be able to handle an increase in problem size or complexity without a significant increase in computation time or increase in complexity to the designer to apply the method. A designer wishing to formulate a new problem from the same design problem domain would be able to retrieve different answers to the same query using different samples from the database. They would be able to see how different designers have conceptually "grouped" variables and functions as semantically related to each other. For example, the multi-objective formulation of the problem also uses monotonicity analysis as the design solution method, but because of the differences in formulation, groupings of variables and functions in design "cases" appear different from the single objective formulation. The method captures these differences, and our results show similar groupings of design cases as reported in the two original sources.

Figure 12

The method is also able to capture invariance of design knowledge within the multiple patterns, because the answers returned for the two different cases have overlaps. This implies, for example, that due to the physics of the system being modeled, if *i* and *s* are semantically related, then the

method applied on both the formulations should return this as a case. If we compare the 2 example cases, to the query variable *i*, the single objective formulation returns the set {*t*, *s*, *p*} as related variables, while the multi-objective formulation returns the set {*t*, *f*, *p*, *s*}. Normally, there will be subjective influences in problem formulation – different designers may choose to model the same problem in different ways. However, there may be some invariant design knowledge that will be common across these different formulations due to the physics of the problem.

Our results show that, through the capture of multiple patterns (different answers to the same query in different formulations), the method captures the differences in formulation. Through the capture of invariant design knowledge (same answers to the same query across different formulations) the method captures the basic "physics" in the formulation.

4.3.2 Effect of the number of retained dimensions

Increasing the number of dimensions *k* shows that the number of variables, parameters, objectives and functions returned reduces, until the final number of dimensions *k* becomes equal to the original rank of the matrix, when it returns exactly the same answers as directly available in the original problem formulation. The reduced dimensionality approximation captures hidden design patterns in the problem statement. For different families of designs, experimentation on various problem sizes will lead to the correct or an optimal number of dimensions that give the best results. Heuristically, the "optimal value" of *k* is the one that leads to well formulated problems. Problem size, complexity of interactions and the design domain characteristics all affect the search for the best value of *k.* For example, in the cylinder design family, due to the relatively small size of the problem, the best dimensions came out to be 2 and 3. Comparing the two formulations, we can see in Figure 12, that in fixing $k = 2$, the multi-objective version (larger problem size) returns more answers (8 or 9) as compared to the answers from the single-objective version (4 or 5). This is an indication that, sometimes, for very large or complex problems, fixing *k* to very low values (say 2) can mean that the algorithm is unable to discriminate between conceptually close pattern groups, and it would need more dimensions to bring out the patterns. Figure 13 shows that for the

multi-objective formulation, the number of answers returned reduces (and becomes more relevant as "correct" answers) as we increase k from 2 to 3. Beyond k=4, the performance deteriorates, and the answers returned are not semantically relevant.

In conclusion, the correct dimensionality is best left as a parameter that the user can modify heuristically for various problem classes and sizes. The problem domain, size and complexity affect the correct value of k. From the mathematics of SVD and dimensionality reduction, it is clear that as the k values approach closer to the rank r (the full approximation) the implied relationships retuned reduce, till by k=r, only the explicit occurrence matrix relationships are returned. Therefore, as a heuristic for choosing a "good" k value, the search is limited to the range of initial k values that return implied relationships when these relations are not evident from the original matrix **A**. To identify the well-formed reformulations, we choose the k values that returned implicit information, and then used the K-means clustering algorithm or the matrix reordering approach based on cosine measurements. Within this range of chosen k values, if a well-formed reformulation exists, both these methods return such a reformulation in the form of tightly coupled clusters of variables and functions.

Figure 13

5 Problem Reformulation Tasks: Aeroengine problem

We now demonstrate the method on a non-analytical Design Structure Matrix (DSM) representation of a large commercial aircraft engine (Pratt and Whitney PW4098) (Sosa et al., 2003) for the following tasks: (1) design decomposition; and, (2) modular and integrative systems analysis. This problem, in demonstrating the design decomposition task, also served to validate whether it is able to identify the 'correct' sub-problem clusters, given a problem with a significant amount of coupling and for which the sub-problem decompositions are already pre-defined by a designer. In the design decomposition experiment, we perform a reverse analysis – we already know the 'answer', i.e. the sub-problem clusters that are pre-defined by the designer, and we check whether the method is able to return the same 'answer'.

5.1 Design Decomposition

Figure 2(b) shows the 8 subsystems that form the aeroengine. The system and sub-system definitions have been identified by a team of collaborating designers and industry design experts (Rowles, 1999; Sosa et al., 2003). A cross between two components in the matrix shows that they share a design interface or dependency. Thus, in the occurrence matrix **A**, A_{ij} is 1 if components *i* and *j* share a design interface, and is 0 if they do not. All diagonal entries A_{ii}=1 with the interpretation that a component shares a design interface with itself. The 54×54 matrix in Figure 2(b) shows the sub-systems already identified. This serves as a validation test – if we merge the sub-system information into a single occurrence matrix, without telling the method about the sub-system definitions that design experts have pre-decided, will the method be able to identify these sub-systems as tightly bound clusters? Thus, the method does not know about the sub-systems, only the explicit design dependency information between any two components. Elsewhere, we have shown using small examples (Sarkar et al., 2008) that the method can be successfully used for design decomposition. In this problem, we test the method on performing design decomposition on a large-scale complex problem.

We applied the method to the matrix **A**, without providing it any identification on sub-system boundaries. The results show that our method is able to identify the sub-system boundaries, keeping a cosine threshold of 0.7, and at k=2. We use Simon's general definition of a "nearly decomposable system" in which the decomposition should lead to weakly interacting subsystems with strong interactions within each sub-system. In terms of our method, the "strong interactions within each sub-system" clause implies that components of the same sub-system have strong local interactions through shared design interfaces, and will show mutually high cosine measurements. This enables them to be identified as a cluster or chunk. We identify these clusters by using the matrix reordering algorithm on cosine measurement matrices (Figures 14 – 16) that cluster together design elements with high similar cosine values, thereby allowing the identification of the cosine threshold.

Figure 14(a-h) shows the cosine measurements for the 8 subsystems. For ease of visual representation, we show these 8 matrices, but they are part of the same large 54×54 matrix produced as a result of the SVD decomposition, and can be identified using the matrix reordering algorithm. Note that very few "outliers" do exist (values lower than 0.7). They point to the observation that two specific components within a sub-system do not share a strong local interaction. This is not unlikely – not all components will have strong local interactions with all the others to be defined as a sub-system. The sub-systems are identified based on the observation that most measurements within a cluster show cosine values well above 0.9.

Figure 14: (a), (b), (c), (d), (e), (f), (g), (h)

Because this method works by measuring distributed "global" patterns of associations between the aero-engine components, it reveals implicit indirect associations as well as explicit ones. Even those components that do not interact directly with each other show a high cosine similarity with each other based on common interaction with a third components. As a simple example, note from the original matrix (Figure 2(b)) that, in the Fan sub-system, component 1 shares design interfaces with components 3 and 4 but not with 5. Both components 3 and 4 share a design interface with 5. Now, note that in the original matrix **A**, there would be a 0 for matrix entry A_{15}. However, the cosine between component 1 and 5 after the *k*-reduction comes out to be 0.9948 (Figure 14(a)), which is an indication showing that the two share a high semantic relation.

5.2 Modular versus integrative systems analysis

Going back to Simon's definition of a "nearly decomposable system", there is another clause – that of "weakly interacting sub-systems". Not only will the components within a sub-system show high interaction, as a group, this sub-system will show low interaction with other sub-systems. Thus, ideally, components within the sub-systems we have identified should show low cosine measurements with all the other components from the other sub-systems. We have demonstrated this for small scale problems (Sarkar et al., 2008). In large-scale design problems such as this one, it is rare for a system to be perfectly decomposable. One of the main problems is that it becomes

difficult to identify sub-system boundaries, because elements within a sub-system share design interfaces with elements from other sub-systems. It is for this reason that we aim for a "nearly decomposable system" rather than a "perfectly decomposable one". It becomes important now to analyze which sub-systems within this large system are modular and which ones are integrative, to reach upon a final decision on the decomposition.

The original source paper provides the following definitions for modular and integrative systems: "a hypothetically perfect modular system would be one whose components do not share design interfaces with components that belong to other sub-systems" On the other hand, a ''hypothetically perfect'' integrative system would be one whose components are completely physically distributed throughout the product resulting in components that share interfaces with all the systems that comprise the product." For a perfectly modular system, our method would show very high intra sub-system cosine measurements and very low inter sub-system cosine measurements.

For the aero-engine problem, however, 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 sub-systems 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. 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 sub-systems.

Following up this example in terms of our method, the LPC system should show high cosine similarity with the Fan and the HPC systems but low cosine similarity with the other systems. Figure 15(a-d) shows 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 pair lead to two different matrices because the original matrix contains asymmetric design relations.

Figure 15: (a), (b), (c), (d)

Figure 16 shows, as examples, that the LPC-HPT and HPT-LPC matrices show low cosine similarity as confirmation that the LPC sub-system shows low cosine similarity with the elements from all other sub-systems for which it does not share design interfaces with. From our results, the LPC sub-systems show low cosine similarity with HPT, LPT and CC systems.

Figure 16: (a), (b)

The external and controls sub-system is identified as an integrative subsystem as our results show that it has high cosine measurements components from almost all the other sub-systems. Our method identifies the Fan, LPC, CC, HPT and LPT as the modular systems. The HPC system falls somewhere in between the clearly modular and clearly integrative systems – it is more modular than the externals and controls system, but more integrative than all the others. These results match the results reported by the authors in the source paper.

In the source paper, the decomposition of the aeroengine into the 8 sub-systems was based on information collected from design experts. This information was not provided to our method, but the decomposition suggested by the method "matches" the results produced by human experts. Decomposition of large systems into sub-systems is heavily dependent on subjective choices exercised by designers. Frequently, there is no one "right" solution. Using this method, and varying cosine thresholds and k values, designers can use this method to observe multiple decompositions. Lastly, because the method captures implicit and explicit relationships between design elements in a distributed way, changing just one dependency value in the original matrix will produce changes in all the resulting matrices and cosine values, which may then alter the decomposition decision suggested. This allows designers to experiment with the matrix entries, i.e. the dependency matrix in an efficient way to observe how the decomposition decision may change.

6 Theoretical conjectures

Design problem modeling and reformulation is not a well-understood cognitive task. How do humans infer and reason in an ambiguous and ill-structured knowledge domain such as design, represent such knowledge using a formal, precise, unambiguous representational system such as mathematics, and juggle between the semantics of design knowledge and syntax of representation in a robust manner? A symbolic AI view would argue that the process is based on rules operating on symbolic representations "all the way down". However, the method we present here shows at least four abilities that are not explained by this view: (1) an ability to infer design semantics that are difficult to encode as formally stated algebraic, logical or symbolic "rules"; (2) an ability to learn implicit semantics that are not explicitly coded into the design representation; (3) an ability to automate a range of tasks that traditionally require a range of different solution algorithms, i.e. different symbolic approaches and methods; and, (4) an ability to infer multiple "right" answers instead of one "right" answer. This last ability also characterizes a quality that separates design from traditional AI problem solving – the existence of multiple non-dominated solutions that can qualify as "right" solutions in an open solution space that develops along with the search for a solution, instead of a single solution or a closed set of solutions, the space of which is defined and fixed at the time the problem is defined. The method shows that it is possible to perform problem reformulation tasks using a pattern recognition and extraction approach on symbols. This is different to symbols being used to define "explicit" rules to reify semantic knowledge in design representations.

Although we do not claim to present a computational model for any cognitive phenomena in this work, we do believe that the structure of cognitive neurobiological mechanisms can provide metaphors to inspire the design of robust computational mechanisms. We show that empirical findings on brain processes in cognitive neuroscience, and theoretical claims for knowledge representation and memory process models in situated cognition and constructive memory viewpoints in cognitive science corroborate the structure and performance of our method. We

present a set of theoretical postulates that propose a perspective on how symbols may interact to account for the demonstrated success of the SVD-based method presented in this paper.

6.1 What is the connection between design syntax and semantics?

Whether knowledge in the brain 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. Design is a discipline with the pragmatic aim of transforming the world we inhabit into a more desirable future world (Simon, 1975; Gero, 1990). Designing, as a process, deals with a defining a formulation that has to be represented in order to be realized, and yet cannot be defined precisely because it does not yet exist. It is recurrently forced to confront conceptual debates on knowledge and its symbolic reification. The physical realization of any design enacts out the tension between the need to reason with abstract, ambiguous, ill-structured (Simon, 1975) knowledge of the world and the need to represent such knowledge in a symbolic, precise, well-structured way in order to formalize the processes of designing and production.

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 external symbolic representation. We work with the following general hypothesis: *If the symbol is a syntactic abstraction produced by the brain (internally or externally), 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 design representation and design semantics*. What is this connection?

The basis of pure 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 pure symbolic AI logic, the symbol "*i*" is a reference to a physical quantity in the world being modeled "internal diameter for a hydraulic cylinder". Similarly, symbol "*s*" is a reference to a physical quantity "hoop stress in the cylinder". Symbolic relations and operations exist between these symbols. Inside a computer, there is no notion of symbol to object mapping. The computer needs a human being to interpret that "*i*" means "internal diameter".

Because the mapping is defined by designers, in such "rule" representations, there is a one-to-one mapping between symbol and meaning. However, the 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 brain. Symbolic operations between "*i*" and "*s"* would be the same as inside a human brain and in a computer. We surmise that there must be a difference based on the evidence that humans can produce design formulations and reformulations, but cannot always explain the "rules" by which they do it. Computers, on the other hand, cannot perform this behavior at all; else, it would have been a routinely automated one.

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, the idea of a direct "symbol" to "object in the world" relationship (the original design occurrence matrix **A**) cannot explain why the design representation should be able to encode "implicit" or "latent" meaning. Why is SVD, which measures syntactical associative patterns, able to reveal implicit semantics that are not explicitly coded into the syntax? Indeed, why did we choose SVD as the algorithmic basis for this method?

As the work in this paper shows us, semantic design knowledge is richer than what is directly observable from the sparseness of its external representation. For example, consider that *i* shares a

"deep", "implicit" or "latent" relationship with the hoop stress equation "s-S<=0", without even occurring in it explicitly. It is hard or impossible to engineer a "symbolic [reasoning] process" based on "comparing them for identity or difference, and acting conditionally on the outcomes of the tests of identity" (Simon, 1995, p.104) to explain such implicit relationships. However, human designers seem to use such associational implicit relationships abundantly, almost in an intuitive, pattern/ perceptual recognition sense. Evidently, there is no one-to-one mapping or correspondence between the symbol and its semantic meaning in terms of the reference to the "thing" in the world out there, between design syntax and design semantics. The semantic meaning contained in a single symbol seems to be distributed over the whole system of symbolic representation, and seems to require this whole system to explain its intended meaning. The one-to-one mapping representation (the original design occurrence matrix **A**) is sparse and the actual frequency of association between a symbol (design element) and its "meaning" within a behavioral relationship (mathematical function, simulation relation, etc.) is extremely low. The simple mapping hides, in a latent way, the richness of associations and correspondences of a different kind.

6.2 "*Symbols aren't simple*": Insights from cognitive science and cognitive neuroscience

Clancey explains this dilemma by explaining 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 *explicit*, *fixed* 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. The same symbol can be a different symbol in different circumstances. In short, 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 continues to change as the conceptual categorizations that lie at its basis keep changing. 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. We summarize the important point that, in a human being, a symbol is not an isolated atomic entity, but is an abstraction that requires a large body of perceptual and conceptual activations to define it. These develop through experience. It is the dynamic relations activated in experiences that give rise to symbols, and these very same relations 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 a "rule" "*s-S<=0"* to form, we first need a dynamic activated relation between *s* and *S* based on all previous experiences that the reasoning entity 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.

So, if symbols result from semantic perceptual conceptual associations, then what can the associations between symbols in design representations tell us about this semantics? The development of expertise in any knowledge domain, including design (Cross, 2004), depends on the ability to abstract and generalize patterns or "chunks" of knowledge over and above the structural details of experiences. Experts are able to extract the underlying principles of a design domain from specific experiences. They show a capacity to develop "deep" semantic knowledge from the structural aspects of the problems that they see. This gives an expert a capacity to re-structure and re-construct this knowledge while applying it in novel situations. Further, experts

demonstrate top down and breadth first strategies in constructing solutions to design problems – an ability that directly shows that given the same set of design requirements, and similar representational systems, experts pursue a set of multiple solutions. In contrast, novice behavior is characterized by a depth first approach, where given one set of design requirements, they explore a single solution in detail before considering another one. This capacity to re-define the problem and consider multiple solutions is typical to design expertise.

Empirical studies in cognitive neuroscience (Mesulam, 1998) show that this is a fundamental ability in human reasoning – the ability to extract invariant features from episodic experiences that occur through perceptual conceptual activations and generalize them into abstract concepts,. The abstracted "knowledge" is then independent of the specific details of these episodic experiences. Further, "in translating sensation into action…identical sensory events can potentially trigger one of many reactions, depending on the peculiarities of the prevailing context" (Mesulam, 1998, p. 1014). Abstraction and generalization of multiple "right" patterns, therefore, seem to be fundamental processes that exist across all ranges and levels of representation. They lie at the basis of the ability to act in a new situation on the basis of what is abstracted and generalized over previous experiences. Carrying this interpretation over to design representations, the SVD method reveals that there can indeed be multiple patterns encoded within a single design representation. There may be multiple potential non-dominant reformulations are possible depending upon the level of abstraction (cosine thresholds and *k*-values). We discussed in Sections 4 and 5, for example, that different design decomposition results appear by varying the cosine threshold and the *k*-value, and all of these may be considered to be "valid" decompositions.

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. Symbolic reasoning and representation is the hallmark for human beings. It is what sets them apart from other species. To understand how symbols may reify knowledge in design representations, we

should explore what kind of abstraction, extraction and encoding of knowledge happens across experiences that involve symbols in large part, such as design experiences.

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, an index is an indication of some correlation, while a symbol is an agreed upon conventional relationship between two things. In this sense, an indexical association is one that exists between a symbol and the object it is referring to. A symbolic association is one that exists between two symbols. He proposes that reference is hierarchical – to be capable of indexical reference is to be already capable of iconic reference, to be capable of symbolic reference is to be already capable of indexical reference. 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) In our case, the symbol is the design variable, and the event or object it is paired to is another variable or 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 internal diameter of a cylinder) even though there is no direct physical referent present (an actual hydraulic cylinder does not exist yet), because "the possibility of this link is maintained implicitly in the stable associations" between symbols. This is also known as decontextualized meanings, a behavior known only to exist in humans and is a key aspect of human symbolic processing. There is a kind of dual reference in operation – symbols refer to objects (sense); they also refer to each other (reference). Finally, after a symbolic system of representation has been developed through extensive indexical interactions between symbol and object, the mutual reference between symbols is used to pick out the reference between objects and

not vice versa. 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 (We refer to our analogy from Section 2 in 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]."

6.2.1 Why SVD reveals design semantics from syntax?

We are now in a position to explain the answer to the "why" question. Why is SVD able to reveal semantic relationships from the syntax? We believe the success of SVD for this method is based on the observation that the syntax (and more explicitly, the occurrence matrix) is a representation in which each symbol occurs contextually with other symbols, and in representing knowledge as such, is a distributed map of associations, relations and patterns between symbols. SVD is able to capture this kind of mutual referencing between symbols that encodes semantic meaning. No single higher order logical/ pure symbolic AI "rule" is able to describe such distributed, multiple, combinatorial referencing. SVD can because it is a unique matrix factorization method that de-couples the distributed associative relationships containing dependency and redundancy information into two independent abstract spaces, and then uses combinations of these abstract vectors to describe the original elements. The dimensionality reduction step in the method re-

structures these relations in a lower dimensional space to bring out the implied strong and weak patterns. Drawing from the above discussion, we propose the following theoretical postulates relating the syntax of design representation, the semantic meaning it intends to represent, and knowledge reification mechanisms that connect the two:

1. A symbolic-mathematical design formulation embeds the semantic meaning of a design object in explicit and implicit ways.
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).
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.
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.

Based on the above postulates, the capacity of SVD to extract implicit semantics and semantics that are difficult to be coded as an explicit "rule" can be explained – we do not need an explicitly stated algebraic rule to capture the knowledge that a design variable is semantically implicitly related to a behavioral constraint in which it does not appear directly. Semantic knowledge about design structure and behavior pertaining to the object in the world is recoded efficiently within a few symbolic associations. Operations revealing these symbolic associations will be able to predict and communicate the actual behavior of design objects in the world, or imaginary ones that do not exist yet. We can now also explain the fact that, as these associations become grounded over different episodic experiences, the probability of choosing a reformulation in the next similar experience is based on a re-structuring and re-representation of older maps of distributed associations.

Semantic meanings are polysemous and contingent upon enacted meanings (the meaning that is possible given experiences and current perception). A single symbol can influence the meaning of other symbols based on the statistical pattern of their occurrence. What SVD does is calculate how the variation of the occurrence of a symbol affects the expression of that symbol and the functioning of that symbol in producing a semantic meaning. An addition to the symbolic "rule based" perspective, AI could incorporate another lower level "rule" – the simple, distributed associations that exist between symbols and their co-occurrence patterns analyzed from a statistical pattern extraction perspective can reveal multiple semantic meanings in design representations.

7 Conclusions

We presented a singular value decomposition based, dimensionality reduction and clustering based method for acquiring semantic design knowledge from the syntax of design representation. The knowledge acquisition problem was modeled using an unsupervised pattern recognition and extraction perspective. We demonstrated the method on a range of problem domains, representational forms and problem sizes. The method performs well in terms of computational efficiency – it is knowledge-lean and does not need high level knowledge engineering; it is training-lean and needs only one or two "training" examples to induce design reformulation decisions. The method demonstrates other abilities that cannot be measured by computational efficiency measurements alone. It is able to extract implicit semantic design knowledge from a representation, is able to extract knowledge that is hard to encode as explicit "rules", is able to extract multiple "right" reformulation decisions from the same design representation, and is able to automate various problem reformulation tasks that traditionally require a range of different algorithms in a simple and efficient manner. In contrast to knowledge-rich AI approaches, this method is based on an unsupervised pattern recognition perspective. While knowledge-driven strategies and techniques make a design computation system powerful and rigorous, and guarantee optimality of decisions, 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. 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 knowledge-rich AI methods and techniques are combined with numerical optimization techniques), it may be beneficial to have exploratory modeling and reformulation methods that allow a designer to develop insight and heuristically explore the problem for symbolic reformulation tasks at the pre-optimization stage.

To explain these demonstrated successes, we explored a possible theoretical basis of why SVD is able to reveal design semantics from design syntax in such a robust manner. An analysis on the nature of symbolic reasoning in human beings deriving from empirical findings and theoretical insights from cognitive science and cognitive neuroscience was presented. The behaviors of the SVD method were compared with these arguments. This resulted in a set of postulates that encode an alternate perspective on how semantic meaning may be encoded in symbolic design representations in a distributed manner, as patterns of associations between symbols and their co-occurrence patterns. These patterns of associations between symbols, re-represented and re-constructed on different abstraction levels, reveal the encoded semantic meaning.

References

Campbell, M.I., Cagan, J., & Kotovsky, K. (2003). The A-design approach to managing automated design synthesis. *Research in Engineering Design 14(1)*, 12-24.

Clancey, W. (1997). *Situated Cognition: On human knowledge and computer representations*. Cambridge University Press.

Clancey, W. (1999). *Conceptual Coordination*. Lawrence Erlbaum.

Cross, N. (2004). Expertise in design: An overview. *Design Studies 25(5)*, 447 - 441.

Deacon, T.W. (1997). *The Symbolic Species: The co-evolution of language and the brain*. Allen Lane (Penguin), London.

Dong, A. (2005). The latent semantic approach to studying design team communication. *Design Studies 26(5)*, 445-461.

Duffy, A.H.B., & Kerr, S.M. (1993). Customised perspectives of past designs from automated group rationalisations. *Artificial Intelligence in Engineering 8*, 183-200.

Ellman, T., Keane, J., Banerjee, A., & Armhold, G. (1998). A transformation system for interactive reformulation of design optimization strategies. *Research in Engineering Design 10(1)*, 30-61.

Gelsey, A., Schwabacher, M., & Smith, D. (1998). Using modeling knowledge to guide design space search. *Artificial Intelligence 101(1)*, 35-62.

Kalman, D. (1996). A singularly valuable decomposition: The SVD of a matrix. *The College Mathematics Journal 27(1)*, 2-23.

Landauer, T.K., & Dumais, S.T. (1997). A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction and representation of knowledge. *Psychological Review 104(2)*, 211-240.

Li, C., & Li, S. (2005). Analysis of decomposability and complexity for design problems in the context of decomposition. *Journal of Mechanical Design 127(4)*, 545-557.

Liu, L., Hawkins, D.M., Ghosh, S., & Young, S. (2003). Robust singular value decomposition analysis of microarray data. *Proccedings of the National Academy of Sciences 100(23)*, 13167-13172.

Medland, A.J., & Mullineux, G. (2000). A decomposition strategy for conceptual design. *Journal of Engineering Design 11(1)*, 3-16.

Mesulam, M.M. (1998). From sensation to cognition. *Brain 121(6)*, 1013-1052.

Michelena, N.F., & Agogino, A.M. (1988). Multi-objective hydraulic cylinder design. *Journal of Mechanisms, Transmissions and Automation in Design 110*, 81-87.

Michelena, N.F., & Papalambros, P.Y. (1997). A hypergraph framework for optimal model-based decomposition of design problems. *Computational Optimization and Applications 8(2)*, 173-196.

Papalambros, P.Y., & Wilde, D.J. (2000). *Principles of Optimal Design*. Cambridge University Press.

Rowles, C.M. (1999). System Integration Analysis of a Large Commercial Aircraft Engine. In, Vol. Masters. Massachusetts Institute of Technology

Sarkar, S., Dong, A., & Gero, J.S. (2008). A learning and inference mechanism for design optimization problem (re)formulation using singular value decomposition. *Proceedings of ASME Design Theory and Methodology Conference*, pp. DETC08-49147. New York.

Schwabacher, M., Ellman, T., & Hirsh, H. (1998). Learning to set up numerical optimizations of engineering designs. *AI EDAM 12(2)*, 173-192.

Simon, H.A. (1995). Artificial intelligence: An empirical science. *Artificial Intelligence 77(1)*, 95-127.

Sosa, M.E., Eppinger, S.D., & Rowles, C.M. (2003). Identifying Modular and Integrative Systems and Their Impact on Design Team Interactions. *Journal of Mechanical Design 125*, 240-252.

Strang, G. (1993). The Fundamental Theorem of Linear Algebra. *The American Mathematical Monthly 100(9)*, 848-855.

Strang, G. (2003). *Introduction to Linear Algebra*. Wellesley-Cambridge Press, Wellesley.

Wolfe, M.B.W., & Goldman, S.R. (2003). Use of latent semantic analysis for predicting psychological phenomenon. *Behavior Research Methods 35*, 22-31.

Learning Symbolic Formulations in Design: Syntax, Semantics, Knowledge Reification

Somwrita Sarkar, Andy Dong and John S. Gero

Figures File

A	d1	d2	c3	c4	c5	m1	m2	m3	m4
human	1	0	0	1	0	0	0	0	0
interface	1	0	1	0	0	0	0	0	0
computer	1	1	0	0	0	0	0	0	0
user	0	1	1	0	1	0	0	0	0
system	0	1	1	2	0	0	0	0	0
response	0	1	0	0	1	0	0	0	0
time	0	1	0	0	1	0	0	0	0
EPS	0	0	1	1	0	0	0	0	0
survey	0	1	0	0	0	0	0	0	1
trees	0	0	0	0	0	1	1	1	0
graph	0	0	0	0	0	0	1	1	1
minors	0	0	0	0	0	0	0	1	1

Figure 1(a): LSA word-by-document example from ((Landauer & Dumais, 1997))

Hydraulic/ Explosive Cylinder Design Problem
FORMULATION EXAMPLE 1

DESIGN VARIABLES:	DESIGN PARAMTERS:	FORMULATION:
i: Internal diameter	T: Minimum Wall Thickness	Min f = i + 2t
o: External diameter	F: Minimum Output Force	Sub to:
t: Wall thickness	P: Maximum Pressure	g1: t - T >= 0
f: Output force	S: Maximum Hoop Stress	g2: f - F >= 0
p: Pressure		g3: p - P <= 0
s: Hoop stress		g4: s - S <= 0
		h1: $f - (pi/4)i^2p = 0$
		h2: s - ip/2t = 0

Figure 1(b): Single objective hydraulic cylinder problem from ((Papalambros & Wilde, 2000))

	d	g1	g2	g3	g4	h1	h2
i	1	0	0	0	0	1	1
t	1	1	0	0	0	0	1
f	0	0	1	0	0	1	0
p	0	0	0	1	0	1	1
s	0	0	0	0	1	0	1
T	0	1	0	0	0	0	0
F	0	0	1	0	0	0	0
P	0	0	0	1	0	0	0
S	0	0	0	0	1	0	0

Figure 1(c): Design occurrence matrix single objective hydraulic cylinder problem

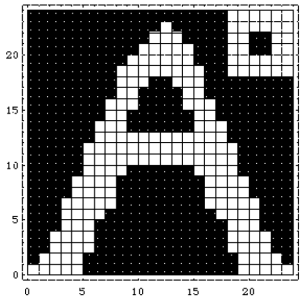

Figure 2(a): a 24×24 image from ((Kalman, 1996))

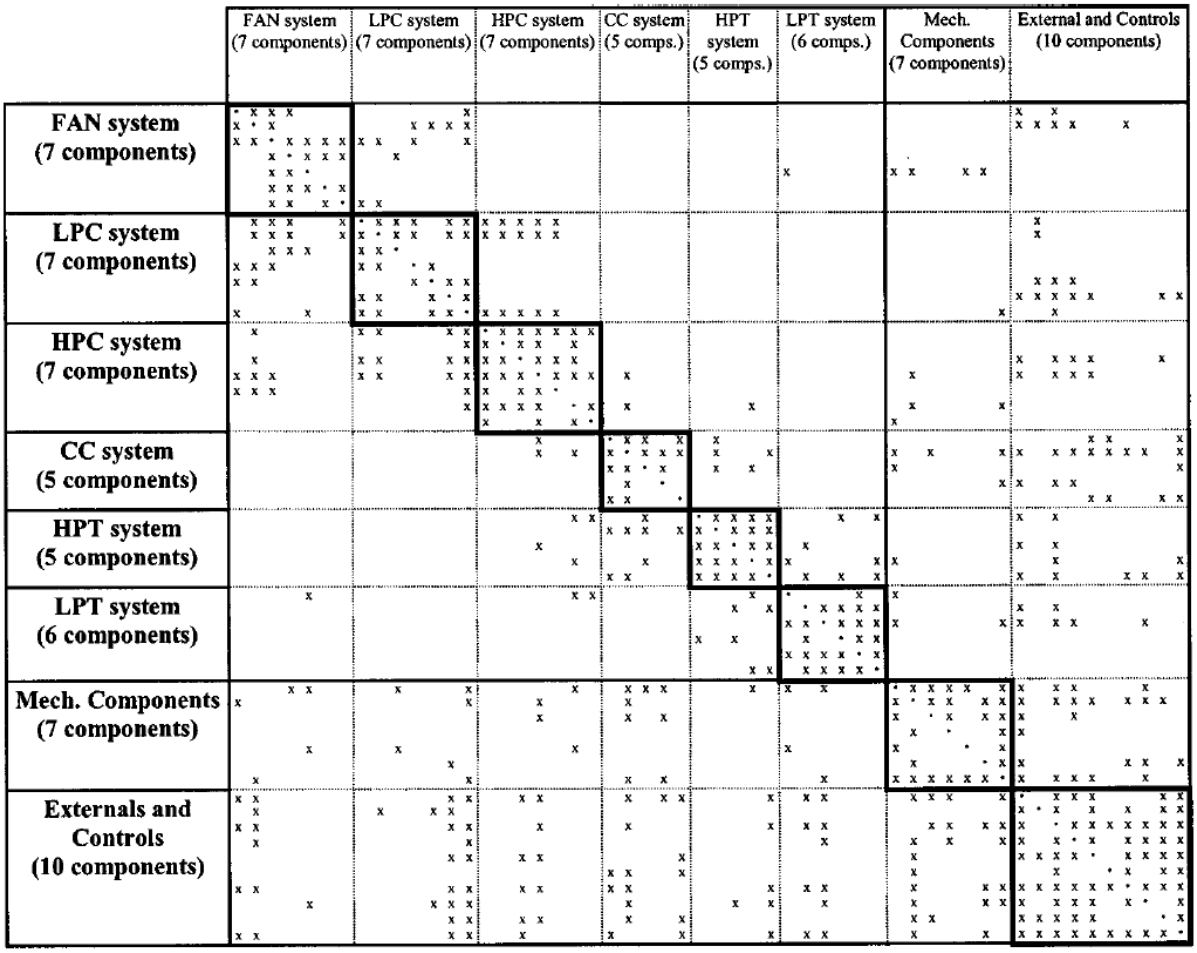

Figure 2(b): a DSM representation from (Sosa et al., 2003))

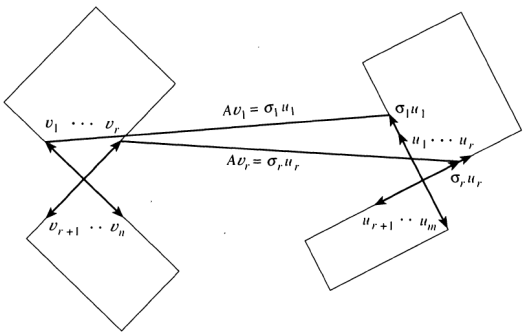

Figure 3(a) (Strang, 1993): Orthonormal bases that diagonalize *A*

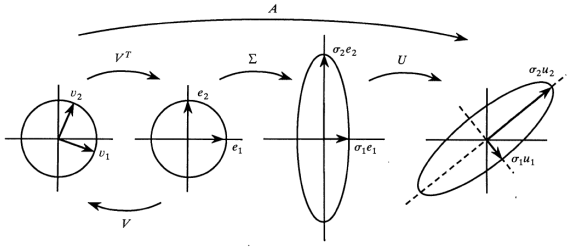

Figure 3(b) (Strang, 1993): Geometrical interpretation of transformation in 2D

$A = USV^{\mathrm{T}}$

U	1	2	3	4	5	6	7	8	9
1	-0.5843	-0.0273	-0.1593	0.0000	0.5443	0.0316	0.3231	0.4718	-0.0902
2	-0.4799	0.5377	-0.3704	0.0000	-0.1980	-0.2621	0.0683	-0.4718	0.0902
3	-0.2142	-0.5892	-0.2842	-0.4083	-0.1117	0.2722	0.0604	-0.3912	-0.3409
4	-0.5214	-0.3596	0.2830	0.4083	-0.1651	0.1488	-0.3322	-0.0806	0.4311
5	-0.3072	0.2296	0.5672	-0.4083	-0.0534	-0.1235	-0.3926	0.0806	-0.4311
6	-0.0759	0.2528	-0.2803	0.0000	-0.5460	0.5588	-0.1032	0.4718	-0.0902
7	-0.0339	-0.2770	-0.2151	-0.4083	-0.3080	-0.5803	-0.0912	0.3912	0.3409
8	-0.0825	-0.1691	0.2141	0.4083	-0.4553	-0.3171	0.5020	0.0806	-0.4311
9	-0.0486	0.1080	0.4292	-0.4083	-0.1472	0.2632	0.5932	-0.0806	0.4311

S	1	2	3	4	5	6	7
1	2.7055	0	0	0	0	0	0
2	0	1.7683	0	0	0	0	0
3	0	0	1.5237	0	0	0	0
4	0	0	0	1.4142	0	0	0
5	0	0	0	0	1.1673	0	0
6	0	0	0	0	0	0.72861	0
7	0	0	0	0	0	0	0.58155
8	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0

V	1	2	3	4	5	6	7
1	-0.3934	0.2886	-0.3477	0.0000	0.2967	-0.3164	0.6730
2	-0.2055	0.4471	-0.4270	0.0000	-0.6373	0.4071	-0.0600
3	-0.0917	-0.4899	-0.3277	-0.5774	-0.3596	-0.4228	-0.0531
4	-0.2232	-0.2990	0.3262	0.5774	-0.5314	-0.2311	0.2919
5	-0.1315	0.1909	0.6539	-0.5774	-0.1719	0.1918	0.3450
6	-0.4879	-0.5520	-0.1054	0.0000	0.2292	0.6212	0.0882
7	-0.6996	0.2151	0.2103	0.0000	0.1095	-0.2817	-0.5733

Figure 4: SVD results for matrix in Figure 1(c)

$U_{[9\times2]} * S_{[2\times2]} * V_{[2\times9]}$

$U_{[9\times2]}$	1	2
1	-0.5843	-0.0273
2	-0.4799	0.5377
3	-0.2142	-0.5892
4	-0.5214	-0.3596
5	-0.3072	0.2296
6	-0.0759	0.2528
7	-0.0339	-0.2770
8	-0.0825	-0.1691
9	-0.0486	0.1080

$S_{[2\times2]}$	1	2
1	2.7055	0
2	0	1.7683

$V_{[2\times7]}$	1	2
1	-0.3934	0.2886
2	-0.2055	0.4471
3	-0.0917	-0.4899
4	-0.2232	-0.2990
5	-0.1315	0.1909
6	-0.4879	-0.5520
7	-0.6996	0.2151

$= A'$

k=2	d	g1	g2	g3	g4	h1	h2
i	0.6079	0.3032	0.1686	0.3673	0.1987	0.7979	1.0957
t	0.7852	0.6918	-0.3467	0.0056	0.3523	0.1086	1.1130
f	-0.0727	-0.3467	0.5636	0.4409	-0.1227	0.8579	0.1814
p	0.3714	0.0056	0.4409	0.5050	0.0641	1.0393	0.8502
s	0.4441	0.3523	-0.1227	0.0641	0.1868	0.1814	0.6688
T	0.2099	0.2421	-0.2002	-0.0878	0.1124	-0.1465	0.2399
F	-0.1053	-0.2002	0.2484	0.1669	-0.0815	0.3152	-0.0412
P	0.0015	-0.0878	0.1669	0.1392	-0.0277	0.2740	0.0919
S	0.1068	0.1124	-0.0815	-0.0277	0.0537	-0.0412	0.1331

Figure 5: *k*-reduced approximation A' of A, $k = 2$

$$U[\text{mX2}] \quad S[\text{2X2}] \quad X[\text{mX2}]$$

$$\begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \\ \cdots & \cdots \\ u_{m1} & u_{m2} \end{bmatrix} \text{X} \begin{bmatrix} s_{11} & 0 \\ 0 & s_{22} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ \cdots & \cdots \\ x_{m1} & x_{m2} \end{bmatrix}$$

$$S[\text{2X2}] \quad V^{\text{T}}[\text{nX2}] \quad Y[\text{mX2}]$$

$$\begin{bmatrix} s_{11} & 0 \\ 0 & s_{22} \end{bmatrix} \text{X} \begin{bmatrix} v_{11} & v_{21} & \cdots & v_{n1} \\ v_{12} & v_{22} & \cdots & v_{n2} \end{bmatrix} = \begin{bmatrix} y_{11} & y_{21} & \cdots & y_{n1} \\ y_{12} & y_{22} & \cdots & y_{n2} \end{bmatrix}$$

Figure 6: Matrix operations for dimensionality reduction, *k*=2

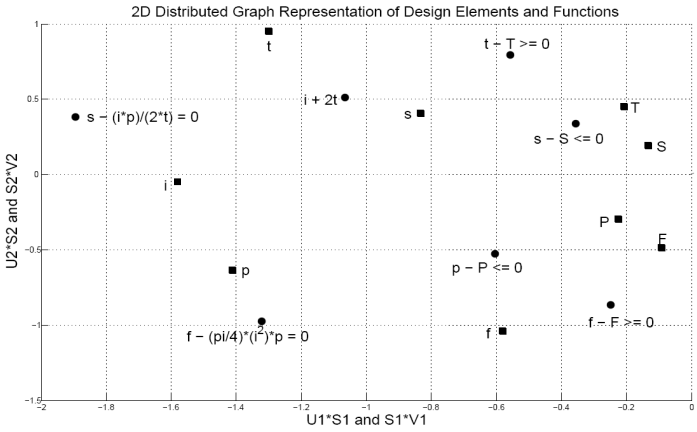

Figure 7: Graph representation for dimension reduction, $k = 2$

Query	i	t	f	p	s	T	F	P	S
i	**1.0000**	**0.7884**	0.5126	**0.9238**	**0.8847**	0.3897	0.2139	0.6224	0.5419

Query	d	g1	g2	g3	g4	h1	h2
i	**0.8881**	0.5500	0.3046	**0.7722**	**0.7041**	**0.8218**	**0.9739**

Figure 8(a): Cosine measurements of *i* with all elements and relationships

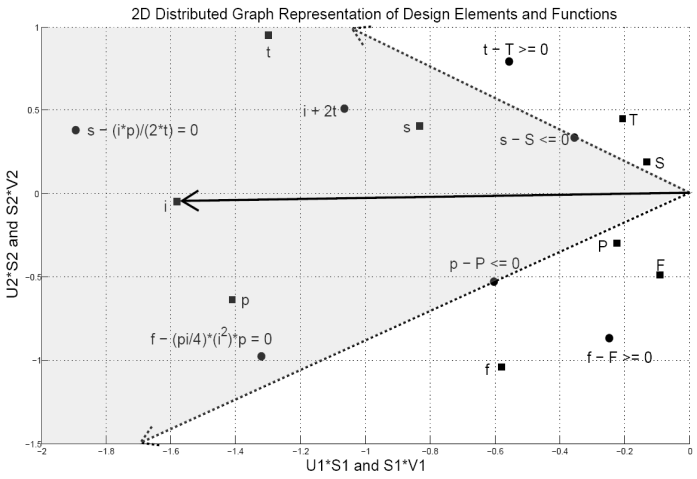

Figure 8(b): Elements and relationships returned for query variable *I*, cosine threshold = 0.7

***k*=2**	***d***	***g*₁**	***g*₂**	***g*₃**	***g*₄**	***h*₁**	***h*₂**
d	1						
***g*₁**	0.8724	1					
***g*₂**	-0.1674	-0.6280	1				
***g*₃**	0.3936	-0.1060	**0.8404**	1			
***g*₄**	0.9518	**0.9803**	-0.4619	0.0925	1		
***h*₁**	0.4679	-0.0239	**0.7930**	**0.9966**	0.1741	1	
***h*₂**	0.9692	**0.7251**	0.0806	**0.6079**	**0.8468**	**0.6711**	1

Figure 9(a): Design "cases" group identification by method

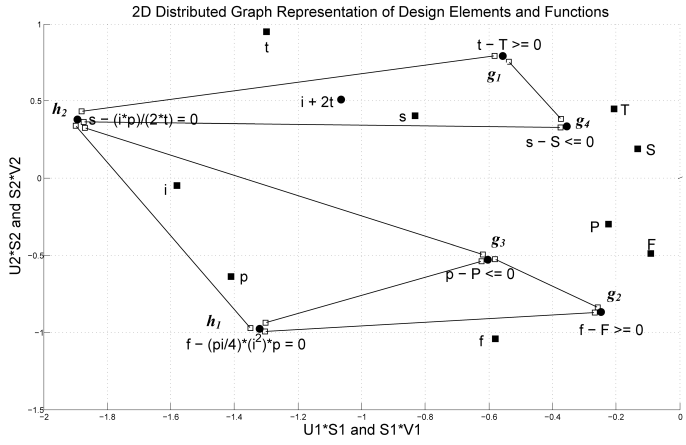

Figure 9(b): Cosine measurements for groups shown in Figure 9(a)

Multi-objective formulation:

Design variables:	**Design model:**
Internal diameter i	Min $Z = z1w1 + z2w2 + z3w3$
Wall thickness t	$z1 = \pi (it + t^2)$
Output force f	$z2 = s/S$
Stress s	$z3 = p/P$
Pressure p	$t - T \geq 0$
Objective Z	$f - F \geq 0$
Objective $z1$	$p - P \leq 0$
Objective $z2$	$s - S \leq 0$
Objective $z3$	$f = (\pi/4)i^2p$
Weight $w1$	$s = p/E\ (i/2t + 0.6)$
Weight $w2$	
Weight $w3$	
Design parameters:	
Min Wall thickness T	
Min Output force F	
Max Stress S	
Max Pressure P	
Joint efficiency E	

Figure 10: Multi-objective formulation of the hydraulic cylinder problem ((Michelena & Agogino, 1988))

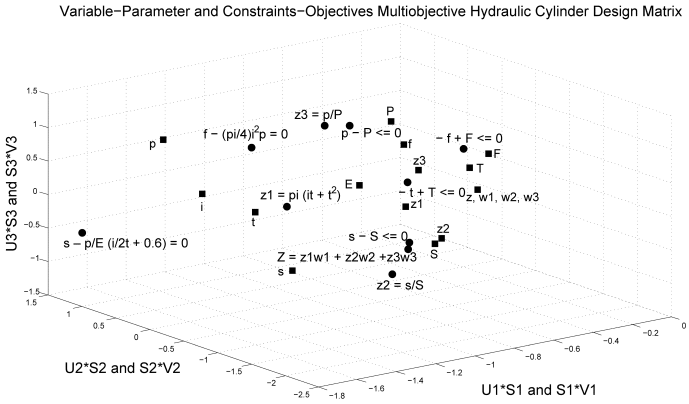

Figure 11: Graph representation for dimension reduction, multi-objective cylinder, k=3

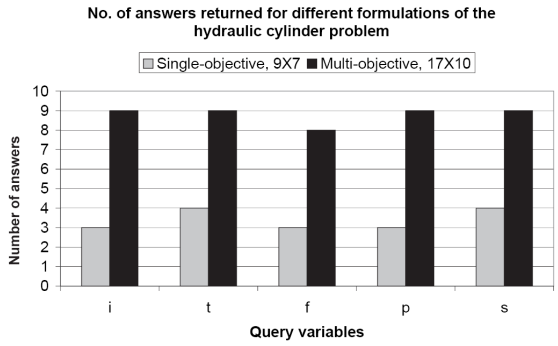

Figure 12: Answers returned from different formulations of the cylinder problem

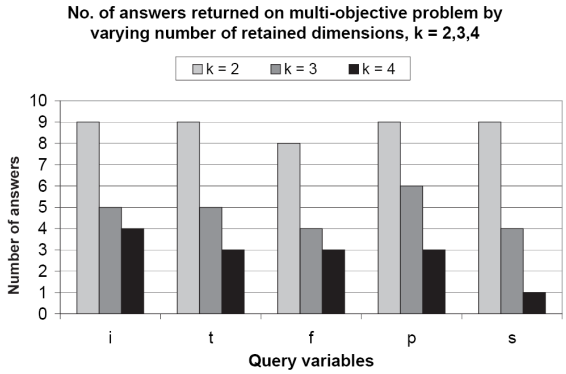

Figure 13: Answers for different k values for the multi-objective cylinder problem

FAN	1	2	3	4	5	6	7
1	0.9968	0.9971	0.8024	0.8261	0.9948	0.6618	0.7017
2	0.9998	0.9842	0.7379	0.7648	0.9793	0.5825	0.6260
3	0.7894	0.8757	0.9983	0.9998	0.8879	0.9646	0.9775
4	0.6527	0.7632	0.9903	0.9837	0.7795	0.9977	0.9999
5	0.9835	0.9430	0.6200	0.6515	0.9342	0.4446	0.4928
6	0.6476	0.7588	0.9893	0.9825	0.7753	0.9981	1.0000
7	0.6307	0.7443	0.9859	0.9782	0.7612	0.9992	0.9999

Figure 14(a): Fan sub-system

LPC	1	2	3	4	5	6	7
1	0.9986	0.9999	0.9995	0.9968	0.8764	0.8172	0.8065
2	0.9986	0.9999	0.9995	0.9968	0.8764	0.8172	0.8065
3	0.9897	0.9833	0.9743	0.9932	0.7659	0.6893	0.6760
4	1.0000	0.9994	0.9969	0.9995	0.8529	0.7892	0.7779
5	0.8215	0.8432	0.8661	0.8055	0.9986	0.9982	0.9970
6	0.6573	0.6863	0.7178	0.6363	0.9556	0.9825	0.9858
7	0.9517	0.9630	0.9740	0.9430	0.9707	0.9378	0.9314

Figure 14(b): LPC sub-system

HPC	1	2	3	4	5	6	7
1	0.9956	0.9961	0.9558	0.8484	0.9933	0.7935	0.9745
2	0.9851	0.9861	0.9760	0.8876	0.9810	0.8392	0.9892
3	0.8896	0.8922	0.9966	0.9843	0.8791	0.9628	0.9881
4	0.8814	0.8841	0.9950	0.9873	0.8705	0.9675	0.9852
5	0.9909	0.9916	0.9671	0.8696	0.9876	0.8181	0.9830
6	0.8114	0.8148	0.9732	0.9996	0.7980	0.9924	0.9540
7	0.9293	0.9315	0.9999	0.9629	0.9207	0.9325	0.9983

Figure 14(c): HPC sub-system

CC	1	2	3	4	5
1	0.9868	0.9947	0.9437	0.9980	0.9975
2	0.9858	0.9953	0.9416	0.9976	0.9979
3	0.9848	0.9047	1.0000	0.9628	0.9180
4	0.9735	0.9994	0.9190	0.9914	1.0000
5	0.9928	0.9894	0.9569	0.9998	0.9936

Figure 14(d): CC sub-system

HPT	1	2	3	4	5
1	0.9440	0.9684	0.9749	0.9925	0.9958
2	0.9959	1.0000	0.9998	0.9924	0.9436
3	0.8851	0.9212	0.9315	0.9637	0.9985
4	0.9712	0.9878	0.9917	0.9996	0.9825
5	0.9623	0.9818	0.9866	0.9981	0.9883

Figure 14(e): HPT sub-system

LPT	1	2	3	4	5	6
1	0.9284	0.9729	0.9918	0.7779	0.7798	0.7760
2	0.9999	0.9908	0.9713	0.9521	0.9530	0.9512
3	0.9947	0.9990	0.9888	0.9205	0.9217	0.9193
4	0.9232	0.8567	0.7982	0.9954	0.9951	0.9957
5	0.9701	0.9238	0.8788	0.9986	0.9987	0.9984
6	0.9666	0.9185	0.8721	0.9992	0.9993	0.9991

Figure 14(f): LPT sub-system

Mech	1	2	3	4	5	6	7
1	0.9563	0.9921	0.9948	0.9947	0.9946	0.9981	0.9995
2	0.9338	0.9984	0.9854	0.9851	0.9850	0.9915	0.9950
3	0.9803	0.9750	1.0000	1.0000	1.0000	0.9993	0.9977
4	0.9881	0.9642	0.9992	0.9992	0.9993	0.9967	0.9938
5	0.9032	0.9998	0.9692	0.9688	0.9686	0.9783	0.9843
6	0.9735	0.9817	0.9994	0.9993	0.9993	1.0000	0.9994
7	0.9543	0.9929	0.9941	0.9939	0.9938	0.9977	0.9993

Figure 14(g): Mechanical components sub-system

Ext/Con	1	2	3	4	5	6	7	8	9	10
1	0.9911	0.8326	0.9981	0.9997	0.9997	0.9321	0.9679	0.9349	1.0000	0.9550
2	0.9294	0.9419	0.9536	0.9644	0.9646	0.8168	0.8785	0.8213	0.9682	0.8548
3	0.9994	0.7737	0.9993	0.9971	0.9971	0.9634	0.9881	0.9655	0.9959	0.9797
4	1.0000	0.7525	0.9975	0.9941	0.9940	0.9717	0.9926	0.9735	0.9924	0.9857
5	0.9733	0.8828	0.9873	0.9926	0.9927	0.8922	0.9388	0.8957	0.9944	0.9214
6	0.9617	0.5417	0.9395	0.9257	0.9254	0.9992	0.9876	0.9988	0.9199	0.9940
7	0.9929	0.8244	0.9989	1.0000	1.0000	0.9373	0.9715	0.9400	1.0000	0.9592
8	0.9986	0.7149	0.9921	0.9866	0.9865	0.9832	0.9978	0.9847	0.9840	0.9935
9	0.9714	0.8866	0.9860	0.9916	0.9917	0.8884	0.9359	0.8920	0.9934	0.9181
10	0.9940	0.8193	0.9993	1.0000	1.0000	0.9404	0.9736	0.9431	0.9999	0.9617

Figure 14(h): Externals and controls sub-system

LPC-Fan	Fan1	Fan2	Fan3	Fan4	Fan5	Fan6	Fan7
LPC1	0.8029	0.8863	0.9967	0.9992	0.8979	0.9585	0.9726
LPC2	0.8029	0.8863	0.9967	0.9992	0.8979	0.9585	0.9726
LPC3	0.6715	0.7792	0.9934	0.9879	0.7950	0.9956	0.9993
LPC4	0.7741	0.8636	0.9994	1.0000	0.8762	0.9708	0.9824
LPC5	0.9965	0.9973	0.8045	0.8281	0.9951	0.6645	0.7043
LPC6	0.9868	0.9493	0.6350	0.6660	0.9409	0.4618	0.5095
LPC7	0.9291	0.9755	0.9424	0.9553	0.9809	0.8530	0.8802

Figure 15(a): LPC – Fan cosine measures

Fan-LPC	LPC1	LPC2	LPC3	LPC4	LPC5	LPC6	LPC7
Fan1	0.8194	0.8413	0.8643	0.8034	0.9984	0.9984	0.9972
Fan2	0.7572	0.7823	0.8090	0.7390	0.9877	0.9990	0.9996
Fan3	0.9996	1.0000	0.9986	0.9984	0.8655	0.8041	0.7932
Fan4	0.9858	0.9784	0.9683	0.9900	0.7495	0.6709	0.6573
Fan5	0.6426	0.6722	0.7042	0.6214	0.9498	0.9788	0.9823
Fan6	0.9846	0.9770	0.9666	0.9891	0.7451	0.6659	0.6522
Fan7	0.9805	0.9721	0.9608	0.9856	0.7303	0.6494	0.6354

Figure 15(b): Fan – LPC cosine measures

LPC-HPC	HPC1	HPC2	HPC3	HPC4	HPC5	HPC6	HPC7
LPC1	0.9998	0.9999	0.9314	0.8068	0.9991	0.7463	0.9553
LPC2	0.9998	0.9999	0.9314	0.8068	0.9991	0.7463	0.9553
LPC3	0.9845	0.9835	0.8427	0.6764	0.9882	0.6025	0.8795
LPC4	0.9996	0.9995	0.9133	0.7783	1.0000	0.7143	0.9404
LPC5	0.8396	0.8427	0.9835	0.9970	0.8271	0.9850	0.9678
LPC6	0.6814	0.6857	0.9095	0.9857	0.6647	0.9973	0.8770
LPC7	0.9612	0.9628	0.9937	0.9316	0.9547	0.8923	0.9992

Figure 15(c): LPC – HPC cosine measures

HPC-LPC	LPC1	LPC2	LPC3	LPC4	LPC5	LPC6	LPC7
HPC1	0.9921	0.9962	0.9991	0.9882	0.9097	0.8576	0.8481
HPC2	0.9790	0.9862	0.9926	0.9730	0.9397	0.8956	0.8873
HPC3	0.8743	0.8927	0.9117	0.8606	0.9989	0.9873	0.9842
HPC4	0.8656	0.8845	0.9043	0.8515	0.9996	0.9900	0.9872
HPC5	0.9860	0.9918	0.9965	0.9810	0.9261	0.8782	0.8693
HPC6	0.7920	0.8153	0.8401	0.7749	0.9948	1.0000	0.9996
HPC7	0.9168	0.9318	0.9469	0.9055	0.9897	0.9675	0.9627

Figure 15(d): HPC – LPC cosine measures

LPC-HPT	HPT1	HPT2	HPT3	HPT4	HPT5
LPC1	-0.4757	-0.3999	-0.3748	-0.2782	-0.0674
LPC2	-0.4757	-0.3999	-0.3748	-0.2782	-0.0674
LPC3	-0.6378	-0.5707	-0.5481	-0.4598	-0.2604
LPC4	-0.5164	-0.4424	-0.4178	-0.3229	-0.1141
LPC5	0.0589	0.1428	0.1697	0.2694	0.4682
LPC6	0.3010	0.3803	0.4053	0.4965	0.6699
LPC7	-0.2337	-0.1510	-0.1240	-0.0220	0.1912

Figure 16(a): LPC – HPT cosine measures

HPT-LPC	LPC1	LPC2	LPC3	LPC4	LPC5	LPC6	LPC7
HPT1	-0.2100	-0.1715	-0.1278	-0.2367	0.3363	0.4394	0.4557
HPT2	-0.4413	-0.4058	-0.3650	-0.4658	0.0973	0.2079	0.2257
HPT3	-0.0638	-0.0246	0.0196	-0.0912	0.4712	0.5668	0.5817
HPT4	-0.3024	-0.2648	-0.2219	-0.3285	0.2448	0.3515	0.3685
HPT5	-0.2691	-0.2311	-0.1879	-0.2955	0.2783	0.3838	0.4006

Figure 16(b): HPT – LPT cosine measures

References

Kalman, D. (1996). A singularly valuable decomposition: The SVD of a matrix. *The College Mathematics Journal 27(1)*, 2-23.

Landauer, T.K., & Dumais, S.T. (1997). A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction and representation of knowledge. *Psychological Review 104(2)*, 211-240.

Michelena, N.F., & Agogino, A.M. (1988). Multi-objective hydraulic cylinder design. *Journal of Mechanisms, Transmissions and Automation in Design 110*, 81-87.

Papalambros, P.Y., & Wilde, D.J. (2000). *Principles of Optimal Design*. Cambridge University Press.

Sosa, M.E., Eppinger, S.D., & Rowles, C.M. (2003). Identifying Modular and Integrative Systems and Their Impact on Design Team Interactions. *Journal of Mechanical Design 125*, 240-252.

Strang, G. (1993). The Fundamental Theorem of Linear Algebra. *The American Mathematical Monthly 100(9)*, 848-855.