Learning Symbolic Formulations in Design Optimization

Somwrita Sarkar and Andy Dong
University of Sydney, Australia

John S. Gero
George Mason University, USA

This paper presents a learning and inference mechanism for unsupervised learning of semantic concepts from purely syntactical examples of design optimization formulation data. Symbolic design formulation is a tough problem from computational and cognitive perspectives, requiring domain and mathematical expertise. By conceptualizing the learning problem as a statistical pattern extraction problem, the algorithm uses previous design experiences to learn design concepts. It then extracts this learnt knowledge for use with new problems. The algorithm is knowledge-lean, needing only the mathematical syntax of the problem as input, and generalizes quickly over a very small training data set. We demonstrate and evaluate the method on a class of hydraulic cylinder design problems.

Motivation

Design formulation and reformulation significantly affect the results of any automated optimization exercise, and is a difficult problem from both computational and cognitive perspectives. It is a difficult problem for many reasons – there is no known formal process that takes an abstract set of design requirements as input and produces a symbolic mathematical design model as output [1]; It is knowledge intensive and requires both domain and mathematical expertise from designers; the numbers of variables, parameters, objectives and constraints in a problem often exceed

Design Computing and Cognition DCC'08. J.S Gero and A. Goel (eds), pp. 533-552. © Springer 2008

human short term memory, with the number of dimensions exceeding human visualization capacities; the problem definition and the design model both change dynamically as the designer's understanding of the problem develops, i.e. the learning and doing of the problem develop together [2]; the solutions produced by the symbolic design model are only as valid as the modeling assumptions that are built into the model, making "reasonable and good" modeling assumptions essential for a valid solution [3]; the adopted modeling formalism is fundamentally related to the kind of algorithms that the designer intends to use to solve the problem. Problem formulation and solution method selection co-evolve [1].

Designers rely upon experience, subjective decision making, and trial and error to produce a final design optimization model that guarantees both the existence of a solution and the finding of an optimal solution through the application of optimization algorithms. Also, designers produce different formulations, given the same design requirements. The process of designing is also a process of discovery and invention, causing the re-definition of the problem along with the construction of the solution [4].

Any learning algorithm assisting the designer in problem formulation would typically be characterized by its ability to learn characteristics of both the designed product and designing process, including both engineering design and representational domain knowledge. As the complexity of engineering design and products increases, computational systems provide support for tasks that were previously considered purely cognition-based human tasks. (Re)-formulation of design problems is one such task. Human designers typically become better at formulating and solving problems as their expertise in a design domain increases with experience. Previous design experiences become the basis of learning; learnt knowledge is applied to future problems, with human designers doing this over much lesser number of experiences as compared to a machine learning system [5]. Computational systems designed for supporting cognitive design tasks must also develop this capacity of learning, generalizing and applying design knowledge using fewer number of design cases as providing a large training database of design cases becomes impractical in real situations.

Aim

This paper presents a knowledge-lean learning and inference mechanism for unsupervised learning of semantic design concepts and relationships from purely syntactical examples of design formulation data in

optimization. It models design optimization problem formulation as a statistical pattern extraction learning problem. Design formulation examples expressed in standard mathematical form, make up the training set. The learning mechanism operates on each sample individually; each sample is defined as one "design experience" for the system. A new experience becomes part of the existing training set. The system then acts as a "design formulation assistant" by extracting the learnt knowledge for new, but similar problems (problems from a common design domain, e.g. the design of hydraulic cylinders, or sharing a common mathematical form, e.g. linearly formulated design optimization problems). The assistance is provided in the form of the designer querying the system for advice. The designer can choose a class or family of optimization problems, define a new problem in this family and then query the system for formulation choices on variables, parameters, constraints and objectives. The algorithm operates as an incremental learning system such that, over time, the same query is able to return different and "richer" answers as the number of "design experiences" provided to the system increases.

The mechanism is based on employing the Singular Value Decomposition (SVD) algorithm from linear algebra [6] [7]. Because the system operates only on syntactical design formulation training data with no other inbuilt knowledge, providing "faulty" data (wrongly or badly formulated examples) can cause the system to learn wrong concepts. However, it may be conjectured that because of the statistical nature of the algorithm, the mechanism should eventually even out "noisy" effects (results from wrongly formulated examples) if enough correctly formulated examples are provided.

The next section presents other related research approaches in the field of machine learning applied to design optimization formulation and reformulation, with comparisons to the method described in this paper.

Related prior work

Many recent approaches in design automation and optimization are based on combining artificial intelligence, machine learning and cognitive science approaches. Balachandran and Gero developed an expert system for rule-based problem formulation [8]. Mackenzie and Gero developed a tool that allows interaction between problem formulations and results [9]. But, in these, the knowledge is not persistent beyond the current design experience, and no new knowledge may be created that can influence

future design tasks. Schwabacher et al. developed a decision-tree inductive learning based optimization tool, where characteristics of the designed product and the optimization process are learnt [10]. Ellman et al. developed a tool allowing the user to formulate, test, reformulate and visualize a tree of optimization strategies constructed by the user that may be applied onto test problems in order to identify relevant optimization strategies for particular design domains [11]. Nath and Gero developed a machine learning tool that acquires strategies as mappings between past design contexts and design decisions that led to useful results [2]. Campbell et al. developed an automated synthesis tool called A-Design. A multi-agent based system combines genetic algorithms, asynchronous teams and functional reasoning to get a team of agents with given roles to evolve conceptual design objects [12]. Moss et al. have further explored learning in such a system, wherein useful "chunks" of design knowledge are learnt and used in future design tasks [5].

Most of the above approaches either require a high level of knowledge engineering (e.g. rules, heuristics, grammars, pre-designed inbuilt algorithms), or a large training database solved of design cases. In this research, we approach the problem from an altogether different perspective – statistical machine learning – inspired in part by research in the statistical natural language processing (SNLP) and digital image processing (DIP) domains. In both these domains, SVD (singular value decomposition) is used as a tool for pattern extraction and dimensionality reduction. In SNLP, SVD based Latent Semantic Analysis (LSA) [13] reveals semantic patterns in textual data. In DIP, SVD [7] is used for identifying pattern redundancy in image data for compression of images. The main insight provided by these methods is that the syntax of representation (words, pixels) embeds the semantics of the knowledge being represented (contextual meaning – sentences, graphic objects), either latently or explicitly. SVD as a dimensionality reduction mechanism, applied in these diverse domains, reveals the hidden or explicit semantic patterns contained in the syntax. We develop a parallel approach, where we use SVD on symbolic-mathematical design formulation data to automate the extraction and learning of semantic design concepts.

Design optimization formulation is a knowledge-intensive task requiring expertise in many domains. To develop learning mechanisms that require high-levels of knowledge engineering or a large training database of cases can be impractical. The main feature of our method is that it is knowledge-lean, and no data or knowledge beyond the mathematical syntax of a problem need be provided as input for learning. Instead of taking a knowledge-based approach, we develop a statistical machine learning

based approach, and conceive the design formulation problem as a pattern extraction problem.

Cognitive-Computational characteristics of design optimization problem formulation

This section presents a discussion on the cognitive-computational characteristics of the problem, its modeling as a statistical pattern extraction problem, the mathematics of SVD, and how the method captures the cognitive-computational characteristics of the problem.

Knowledge representation

For this work, we assume that an optimization problem is formulated in its general canonical form by the designer:

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

$$\text{Sub to } \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{P}) \leq \mathbf{0}, \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{P}) = \mathbf{0}, \boldsymbol{x}, \boldsymbol{P} \in \mathrm{X} \subseteq \mathbf{R}^{\mathrm{n}}$$

Here, $\boldsymbol{x}$ is the vector of design variables and $\boldsymbol{P}$ is the vector of design parameters. Design or decision variables are what the designer can vary in the search for a solution. Parameters are kept fixed for a particular design model, symbolically or numerically. $\boldsymbol{g}$ is a vector of inequality constraints and $\boldsymbol{h}$ is a vector of equality constraints that the design must satisfy to be feasible. Both $\boldsymbol{x}$ and $\boldsymbol{P}$ belong to some subset X of the real space $\mathbf{R}^{\mathrm{n}}$, where n defines the total number of dimensions in terms of the numbers of variables and parameters. The functions $\boldsymbol{f}$ are objectives to be minimized; for a single objective case, this becomes a single function f.

In general, there are many subjective decisions that the designer makes in formulating this model [1]. Examples are choices on which elements are to be decision (design) variables and parameters, whether the design constraints to be incorporated in the model are natural ones, described by the physics of the system, or pragmatic ones, guided by engineering and design guidelines and "rules of thumb", or ones that are imposed externally (manufacturing limitations, material availability, interdisciplinary constraints), which algorithm is chosen to solve the model, how formulation decisions change based on the choice of algorithm, any mathematical features in the problem (e.g. linearity) that require special formulations, and solution algorithms (e.g. linear programming) etc. As is evident, not only do the designers' personal experience based choices form the basis of this decision-making, many

external factors beyond the direct control of the designer come into play as well. This makes the formulation problem a dynamic one. As the understanding of a problem grows after initial formulations, many reformulations are normally required to finally reach a design model that produces a satisfactory result. The mathematical model contains and represents, either implicitly or explicitly, many of these subjective decisions along with the characteristics of the object being modeled.

Cognitive hypothesis as basis of the learning and inference mechanism

Modeling design formulation and reformulation as a statistical pattern extraction problem is based on our hypothesis that the mathematical representation of optimization problems contains, in hidden and explicit ways, many semantic design concepts and relationships. Many of the choices available to the designer as discussed above are embedded in the syntactic structure of the formulation – the compact notation of mathematics. The syntax can be "expanded out" to reveal these embedded semantic relations.

Designers often formulate and reformulate problems because they discover hidden semantic or mathematical relationships between variables, parameters, objectives and constraints – ones that were not directly observable before, or new ones that appear as designing progresses. This helps them to redefine the design problem, re-representing it in a form that becomes easier to solve or captures new design semantics. The main insight is that the problem representation itself captures design semantics in a latent manner, i.e. the variables, parameters and constraints as design concepts occur in "context" of each other, capturing designer intentions and choices and the functional-behavioral-structural relationships of the design object being modeled.

There are close parallels between this hypothesis and work done using natural language processing applied to design. Dong and Agogino use information retrieval methods to explore construction of design representations from textual documentation [14]. Hill et al. apply the Latent Semantic Indexing method for document analysis to show how designers develop a "shared understanding" of a design object that is inherently captured in textual design documentation [15]. According to Moss et al. internal and external knowledge representation mechanisms and changes in them capture the structure and content of a domain, as well as internal cognitive processes [16]. In all these studies, the implicit assumption is that the textual-syntactic representation of a design captures not only the semantic concepts and choices the designer employs, but also

the structural-behavioral-functional characteristics of the design object itself.

Since mathematics is a formal language, it has characteristics very different from natural language. The syntactic-semantic relationships are much more strict and precise with little of the ambiguity that characterizes natural language. However, it is the primary symbolic language employed by designers for constructing representations in design optimization [1]. It, therefore, seems reasonable to draw a parallel and hypothesize that the mathematical representation of a problem, implicitly or explicitly, captures the design object semantics as well as the semantic choices exercised by the designer in producing that representation. Changes in this representation as the optimization process passes through reformulations will reflect how the modeling of the design object changes, as well as how the designer changes his/ her choices and decisions on the design.

In addition to this hypothesis, another important observation from the cognitive viewpoint is that designers typically become better at solving problems based on their experiences [5]. Their performance improves with their experiences in solving problems in a particular domain. Each engineering and design domain tends to develop "favorites" and "best practices" – preferred formulations and preferred algorithms for solving these models. Over time, the discipline, as a whole, discovers what works well and what does not for a particular family of problems [1] [11]. What works is usually based on many factors – the design-based and mathematical characteristics of the problem being modeled, pragmatic engineering and design considerations, or, very often, a designers' developed understanding of a formulation or algorithm developing into a preferred choice.

From the cognition perspective, this provides two lessons for the design of a learning and inference mechanism: (a) Look for patterns of design semantics, concepts, relationships and decisions in the syntactical (mathematical-symbolic) structure of a design formulation, and (b) Base the design of a learning and inference mechanism on statistics, i.e. use examples of different types of formulations of the same problem to learn how these semantic "patterns" evolve for a particular family or class of problems.

Computational basis for the design of the learning and inference mechanism

We use the Singular Value Decomposition (SVD) algorithm from linear algebra in order to perform semantic pattern extraction from syntactic examples of design formulation. SVD is an attractive choice because,

mathematically, it manages to bring out many of the characteristics discussed in the above section in a relatively simple way. In general, SVD takes any general rectangular matrix A with m rows and n columns and decomposes it into a product of three matrices, $A = USV^T$, where U ($m \times m$) and V ($n \times n$) are the orthogonal matrices with columns that are left and the right singular vectors and S ($m \times n$) is a rectangular matrix with singular values on the diagonal that can be arranged to be non-negative and in order of decreasing magnitude. The number of singular values is r, where r is the rank of A. The implicit mathematical idea [6] is as follows. A is an $m \times n$ matrix, whose row space is r-dimensional and inside $\mathbf{R}^n$ and whose column space is also r-dimensional and inside $\mathbf{R}^m$. For this decomposition, we have to choose orthonormal bases $V = (v_1, v_2, \ldots v_r)$ for the row space, and $U = (u_1, u_2, \ldots u_r)$ for the column space. Av_i should be in the direction of u_i, with s_i providing the scaling factor, i.e. we want to choose v_i and u_i such that $Av_i = s_iu_i$. In matrix form, this becomes $AV = US$ or $A = USV^T$. A is the linear transformation that carries orthonormal basis v_i from space $\mathbf{R}^n$ to orthonormal basis u_i in space $\mathbf{R}^m$. The singular values expand or contract – scale the vectors that are being carried from one space to the other.

Any general matrix A where the rows stand for design concepts and elements (variables, parameters) and the columns stand for the "context" of the occurrence of these concepts and elements (mathematical functions) can be decomposed into independent principal components (columns of U and V) represented by a set of new, abstract variables that represent correlations between the variables of the original data set. SVD analysis on the matrix produces a distributed re-representation of the original data (that can be combined again to generate the original data perfectly), where the new abstract variables produced contain within themselves information with regard to all correlations between the original matrix entries. If one entry in the matrix is changed, then this is enough to produce changes in all of the components. The same decomposition can also be used to provide an approximation of the matrix A in a linear least squares sense. If in the decomposition $A = USV^T$ only the first k singular values are retained, a new matrix $A\square$ is returned that is an approximation of the original matrix A. This approximation is a dimensionally reduced approximation of the original matrix A that preserves latent and explicit relationships between design elements and functions. SVD is, therefore, a method for producing distributed re-representations of original data, as well as a method for performing dimensionality reduction. This dimensionally reduced re-represented data is queried by the designer by measuring the semantic "closeness" or "distance" between design concepts through cosine

calculations. This information is useful for use with new problems. Doing this for different formulations of the same problem captures characteristics from different representations of similar classes of optimization problems.

Detailed methodology, demonstration and results on a hydraulic cylinder design problem

Representation

The data representation used draws an analogy with natural and formal languages. It assumes that variables and parameters are elements (symbols) that occur in and come together by rules (syntax) of mathematics to form functions as constraints and objectives (clauses). We take formal optimization problem statements and convert them into occurrence matrices that measure whether each element (variable, parameter) occurs in each function (objective, constraint). The rows of the matrix A represent design elements (variables and parameters), while the columns represent the syntactic "context" in which design elements occur (mathematical functions – objectives and constraints). Each entry A_{ij} is a number specifying whether or not the particular variable or parameter i occurs in a particular constraint or objective j. This provides the raw data for each case on which SVD analysis is carried out. One such matrix is generated for each "design experience" or one example of a design formulation. The basis for this data matrix is that variables and parameters that share high semantic correlations will tend to co-occur together in the same objectives and constraints (for example, length l and breadth b will always co-occur in an area function $a = l \text{ X } b$).

Data generation

Identify a particular design problem domain

The method developed is general and may be applied to any other problem domain, provided the design problem is stated as a formal optimization model. The design of hydraulic cylinders was chosen as a demonstration example primarily because it has been solved in various ways, allowing us to compare approaches. There is nothing special in this example that affects the results.

Select formal optimization models from books and journal papers

Although design optimization models have a general mathematical form, the specifics of the forms differ. The same problem may be modeled as a single or multi objective problem, with different sets of variables, parameters and constraint functions. We take example problems from various sources and use the authors' formal models to provide the basis for our data. Different symbolic models capture different design intentions. The learning and inference mechanism uses these as training experiences to make predictions on which sets of variables, parameters and functions have high (or low) semantic correlations. This information is helpful in formulating new problems. The new problems when formulated join the existing database, thereby enlarging the experience database with each example. Figure 1 shows an example problem formulation of the single objective hydraulic cylinder design problem [1].

Hydraulic/ Explosive Cylinder Design Problem
FORMULATION EXAMPLE 1

DESIGN VARIABLES:

i: Internal diameter
o: External diameter
t: Wall thickness
f: Output force
p: Pressure
s: Hoop stress

DESIGN PARAMTERS:

T: Minimum Wall Thickness
F: Minimum Output Force
P: Maximum Pressure
S: Maximum Hoop Stress

FORMULATION:

Min f = i + 2t
Sub to:
g1: t - T >= 0
g2: f - F >= 0
g3: p - P <= 0
g4: s - S <= 0
h1: f - (pi/4)i²p = 0
h2: s - ip/2t = 0

	f	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

Fig. 1. The hydraulic cylinder design problem [1] and related occurrence matrix

Generate the variable-parameter-constraint-objective matrix

The variable-parameter by objective-constraint matrix A is comprised of m variables and parameters x_1, x_2, … x_m occurring in n objectives and constraint functions y_1, y_2, … y_n, where the entries A_{ij} indicate whether or not a variable or parameter x_i occurs in objective or constraint y_j. Figure 1 shows the matrix generated for the example problem.

Singular Value Decomposition Analysis

Perform Singular Value Decomposition on the matrix

SVD is performed on the matrix A, producing a distributed re-representation of the data in terms of a new abstract space, the components of which space are independent of each other, but are created by mutual

correlation information contained in the original data matrix. The underlying assumption is that patterns of high and low correlation between variables and parameters will emerge, as highly correlated variables have a tendency to appear together in some constraints and objectives, while negatively correlated variables have a tendency not to. Figure 2(a) shows the SVD decomposition for the data matrix shown in Figure 1.

(a) $A = U\Sigma V'$, SVD decomposition for data matrix in Figure 1

U

-0.58432	-0.027303	-0.15931	-9.4601e-0...	0.54433	0.031583	0.3231	0.47184	-0.090179
-0.47993	0.53769	-0.37041	9.4601e-0...	-0.19799	-0.26214	0.06829	-0.47184	0.090179
-0.21423	-0.58922	-0.28424	-0.40825	-0.11171	0.27223	0.06038	-0.3912	-0.34087
-0.52143	-0.35962	0.28296	0.40825	-0.16509	0.14877	-0.3322	-0.080644	0.43105
-0.30721	0.22961	0.5672	-0.40825	-0.053386	-0.12346	-0.39258	0.080644	-0.43105
-0.07594	0.25281	-0.28027	5.9136e-0...	-0.54599	0.55879	-0.10319	0.47184	-0.090179
-0.033897	-0.27704	-0.21507	-0.40825	-0.30804	-0.58029	-0.091236	0.3912	0.34087
-0.082507	-0.16908	0.2141	0.40825	-0.45525	-0.31712	0.50196	0.080644	-0.43105
-0.04861	0.10796	0.42917	-0.40825	-0.14722	0.26317	0.5932	-0.080644	0.43105

Σ

2.7055	0	0	0	0	0	0
0	1.7683	0	0	0	0	0
0	0	1.5237	0	0	0	0
0	0	0	1.4142	0	0	0
0	0	0	0	1.1673	0	0
0	0	0	0	0	0.72861	0
0	0	0	0	0	0	0.58155
0	0	0	0	0	0	0
0	0	0	0	0	0	0

V

-0.39336	0.28864	-0.34766	0	0.29669	-0.31644	0.673
-0.20546	0.44705	-0.42704	7.4555e-0...	-0.63734	0.40714	-0.06001
-0.09171	-0.48989	-0.3277	-0.57735	-0.35958	-0.4228	-0.053058
-0.22322	-0.29899	0.32623	0.57735	-0.53143	-0.23106	0.29192
-0.13151	0.1909	0.65393	-0.57735	-0.17185	0.19175	0.34498
-0.48788	-0.55203	-0.10539	2.4763e-0...	0.22918	0.62115	0.088172
-0.69964	0.21511	0.21031	-4.9659e-0...	0.10953	-0.2817	-0.57328

(b) *k*-Reduced approximation: k = 2, retained k values = 2.7055 and 1.7683

0.60792	0.30322	0.16863	0.36733	0.19869	0.79793	1.0957
0.78519	0.69183	-0.3467	0.0055679	0.35227	0.10863	1.113
-0.072747	-0.3467	0.56358	0.4409	-0.12268	0.85794	0.18137
0.37139	0.0055679	0.4409	0.50504	0.064139	1.0393	0.85022
0.44413	0.35227	-0.12268	0.064139	0.18682	0.18137	0.66884
0.20985	0.24206	-0.20016	-0.0878	0.11236	-0.14654	0.23991
-0.10532	-0.20016	0.2484	0.16694	-0.081459	0.31518	-0.041219
0.0015083	-0.0878	0.16694	0.13922	-0.02772	0.27396	0.091859
0.10683	0.11236	-0.081459	-0.02772	0.053739	-0.041219	0.13308

Fig. 2.(a) SVD decomposition for data matrix in Figure 1 (b) k-Reduced approximation for original data matrix A, k = 2, retained k values = 2.7055 and 1.7683

Retain the first k important singular values and produce a dimensionally reduced truncated SVD

From this computed SVD, retain the first k important singular values and compute a truncated SVD, which is a least squares approximation of the original one. For the purposes of visualization and the relatively small dimensional sizes of the original problems used for demonstration in this paper, we fix $k = 2$ and 3, i.e. the first 2 or 3 singular values are used to compute a truncated matrix. For larger problems, one may experiment with

more number of dimensions. Deciding the correct number of dimensions to retain for different families of problems is important. It can be answered through experimentation, though from a heuristic point of view the original problem size is a good indicator of the value of k. Generally, the smaller the problem size, the smaller the required k value, and the larger the problem size, the higher the value of the required k. Figure 2(b) shows the k-reduced approximation matrix for k = 2. Note how the original 0s and 1s have changed to higher or lower values – this is the main claim, that a reduced dimensionality re-representation will capture the hidden semantic patterns between design elements and functions, and that these patterns may not be directly observed from the original syntactic formulation.

Use this data to generate distributed graph representations:

It is helpful to use the k-reduced approximation (for k = 2 or 3) to visualize the patterns between the elements and functions. For higher k, the computational mechanism stays the same, although we cannot visualize it. For example, in the 2 dimensional case, the first column of is U multiplied by the first singular value and the second column of U multiplied by the second singular value to produce the x and y coordinates of the design elements (variables and parameters) in 2D space. Do the same with V for obtaining the x and y coordinates for visualizing the design functions (objectives and constraints) in the same 2D space. Plot this data as a 2D graph. For a 3D graph, retain the first 3 values for each. In this dimension reduction and feature extraction process, one obtains a distributed context sensitive representation of the all the original variables, parameters, objectives and constraint functions in a semantic lower dimensional space. This is used to retrieve data for new experiences, as this representation captures semantic patterns existing in the formal syntactic representation. Figure 3 shows the graph for the 2D case.

Querying the data for information retrieval

For new problems, use the old problem graphs to make new predictions. Each experience is stored as one SVD matrix and graph. The following queries are possible:

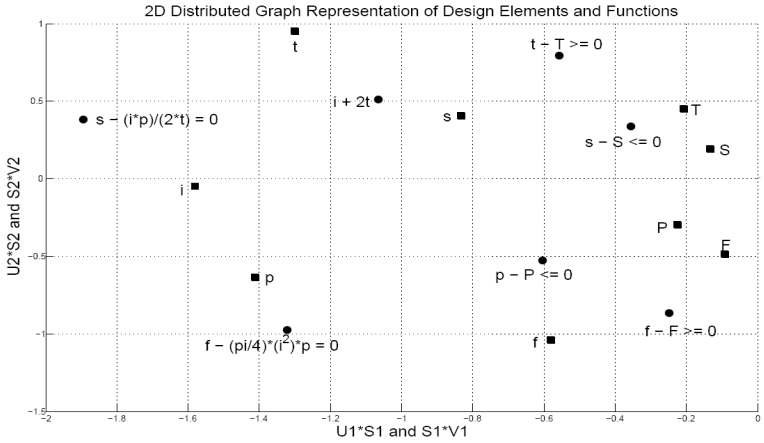

Fig. 3. Distributed graph representation of elements and functions, 2 dimensions; variables, parameters, constraints and objective projected into the same 2D space

Variable-variable or variable-parameter correlations and query retrieval:

This query extracts information on what other variables or parameters should one consider in formulating the problem, given that a variable x has been chosen by the designer. This query is answered calculating the cosine of the angle θ between the query variables and the U^*S vectors. The closer the value of cos(θ) is to 1, the higher the semantic similarity between two variables or parameters. The closer the value of cos(θ) is to 0, the lower the semantic similarity between the two variables or parameters. A threshold value of cos(θ) may be decided upon, such that all variables and parameters having a cos(θ) value higher than this threshold measured with the query variable is returned as the answer of the query for the new problem. Setting the cosine threshold higher or lower will return fewer or more numbers of variables – this is purely a matter of trial and error and there is no set threshold that may be considered as a benchmark.

For example, in the hydraulic cylinder design example problem, setting the query variable as internal diameter i with a cosine threshold of 0.7 returns variable set $\{t, p, s\}$ as other relevant variables. Figure 4 demonstrates this.

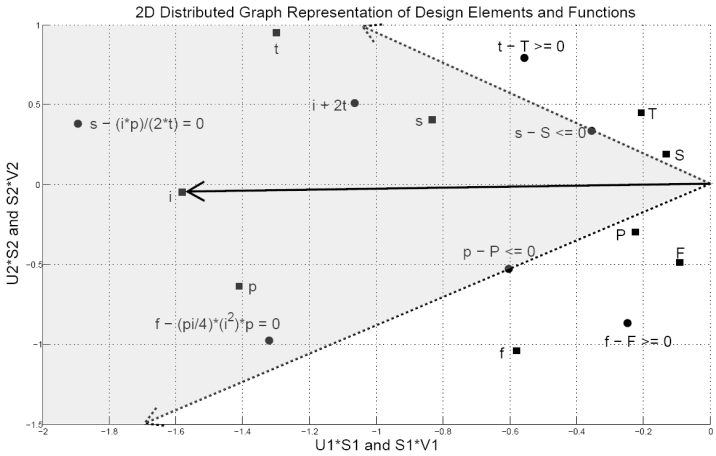

Fig. 4. The element-function set returned for query variable i; Cosine threshold = 0.7; Variable-parameter set returned = {t, p, s}; Function set returned = {i+2t, p-P<=0, s-S<=0; f-(pi/4)i^2p=0, s-ip/2t=0}; (the middle arrow shows the query variable, the side arrows show the limits set by the cosine threshold of 0.7

Variable-function correlations and query retrieval:

A similar cosine similarity measurement is also possible between a variable and a constraint or objective function – i.e. if the designer is considering a particular variable or parameter, which constraints and objectives may be relevant in modeling? A similar cosine analysis, but now between the variable and constraints and variable and objective, will return the relevant constraints and objectives as answer to the query. Figure 4 shows the relevant function set returned by setting the query variable as *i* with the cosine threshold set to 0.7. It is evident from the results that the method returns even those variables and constraints as an answer that do not directly co-occur with the query variable in individual functions. For example, *i* co-occurs with *p* and *s* in the two equality constraints, but does not occur anywhere in the two pressure and stress inequality constraints ($p - P <= 0$ and $s - S <= 0$), but these two are still returned as relevant to variable *i*. This is an interesting result. The SVD method, because it captures all direct and indirect syntactic correlations in a distributed way, reveals even indirect, "hidden" relationships between variables that may or may not occur together in functions explicitly.

Design "cases", constraint activity and inactivity identification:

Consider solving this example problem using monotonicity analysis [1]. This method is well documented on this problem in not only [1], but also other sources [17]. Monotonicity analysis is a problem-solving-by-reformulation method, where constraint activity information is used to reformulate the problem to a simpler form. The method discovers design cases – sets of constraints that can be active or inactive, in order to reach upon simpler forms that may then be easily solved by traditional means (unconstrained optimization etc.). A significant charcateristic to note here is that this process of reformulation is actually similar to discovering previously unobserved "hidden" relationships between variables, parameters and constraints, that when made explicit, make solving the problem easier.

In the case of this example problem, there are 5 design variables, and 6 design constraints. The mathematical theorem for monotonicity analysis states that 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 solution cases. Papalambros and Wilde develop a monotonicity analysis based solution procedure, identifying 3 design cases – stress-bound, pressure-bound, and thickness-bound [1]. 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. Figure 5 shows that the SVD analysis followed by cosine measurements shows similar conclusions by purely a syntactic analysis of design formulation. Observe that constraints (*g2*, *h1*) and (*g4*, *h2*) form distinct visual groups, with constraints *g3* and *g1* falling close to these two groups, a fact numerically confirmed by the cosine measurements.

Results over many design experiences

We applied the method on a family of hydraulic cylinder design problems. There are multiple ways in which the same problem has been formulated by different designers. For example, in [1] it is formulated as a single objective problem (Figure 1) while in [17] it is formulated as a multi-objective problem with conflicting objectives and a slightly different set of constraints and parameters. The SVD based algorithm was applied to these problems. The main conclusion to report is that the algorithm generalizes

very quickly over a very few number of training examples. This is interesting because, in general, machine learning systems for design support require either a very high level of knowledge engineering, or they require a large database of training examples. This method requires only 2 to 3 training examples for it to generalize, while all further examples are "real" design experiences that either reinforce or change the learning gained from these first examples. For example, we used the querying techniques on these 2 experiences [1] [11], we found that the algorithm returns "correct" answers for query variables that exist in the first two examples (for example $\{t, p, s\}$ for i using a cosine threshold of 0.7, with k = 2 or 3). Because the algorithm is based on pure statistical extraction of patterns, it does not discriminate between "wrongly formulated" and "correctly formulated" examples. However, it can be conjectured that over many numbers of design experiences, 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.

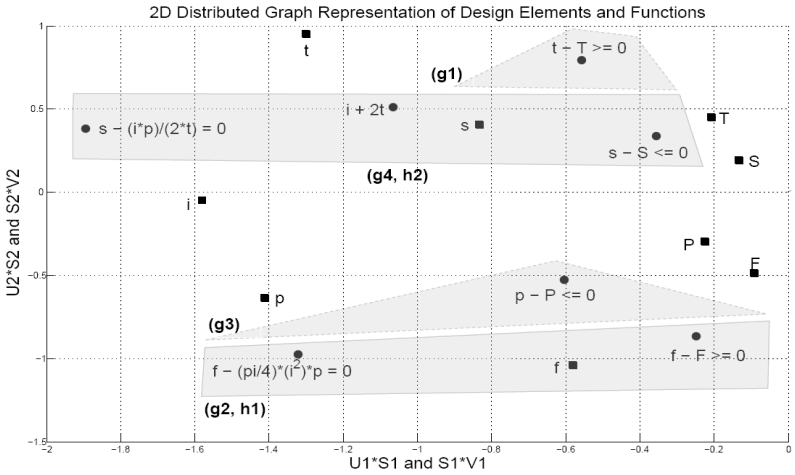

Fig. 5. Design "case" identification using SVD based mechanism for hydraulic cylinder design problem 2D representation.

Specifically, we report the effects of changing two parameters in the algorithm, the size of the design problem with variances in formulation, 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 measured by observations on how each of the answers matches up with those provided by any other relevant and

documented design solution method that actually solves the problem. 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.

Effect of problem size

Keeping the number of dimensions *k* fixed, the algorithm 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 number of variables and constraints than the single-objective version. If we keep the number of dimensions fixed for both design experiences, the number of answers returned to the query variables in the multi-objective case is higher than in the single objective case. Figure 6 shows this result, as over two examples of the same problem with different formulations, we see that the algorithm manages to capture the variances in the formulations. The single objective formulation has an original matrix size of 9 X 7 (Figure 1). The multi-objective formulation of the same problem has an original matrix size of 17 X 10 as the multi-objective problem has an increased number of variables, parameters and objective function and weight identifiers [17]. The algorithm returns a higher number of design concepts as answers in the multi-objective case. This is an interesting result as a designer wishing to formulate a new problem from the same family would be able to retrieve different answers to the same query using different examples 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 SVD method captures this, and our results show similar groupings of design cases as reported in the two original sources.

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 matrix dimensions,

when it returns exactly the same answers as directly available in the original problem formulation. A reduced dimensionality captures hidden design patterns in the problem statement. For different families of designs, experimentation on varied problem sizes will lead to the "correct" number

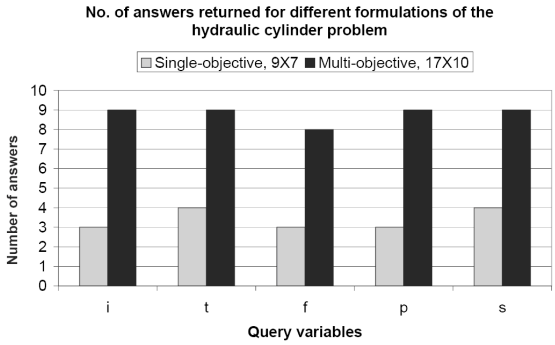

Fig. 6. Answers returned from different formulations of the cylinder problem

of dimensions that give the best results. Heuristically, the size of the original problem is a good rough indicator of the best value for *k*. A small problem size indicates a low value of *k* and a larger problem sizes imply larger values. 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 6, that in fixing k = 2, the multi-objective version (larger problem size) returns more number of 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 problems, fixing *k* to very low values (say 2) can mean that the algorithm is unable to discriminate between finer pattern groups, and it would need more dimensions to bring out the patterns more clearly. Figure 7 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. Again, the "correctness" of answers is measured by the match on results to a published method for solution to the same problem. In the multi-objective case, the documented method was [17]. In conclusion, the correct dimensionality is best left as a parameter that the user can tweak for various problem classes and sizes. The problem domain and the problem size affects the correct value of *k*.

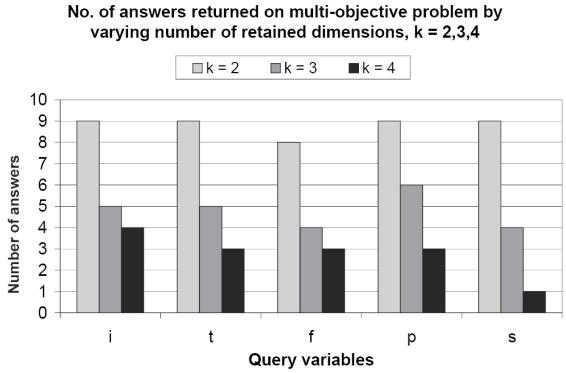

Fig. 7. Answers for different k values for the multi-objective cylinder problem

Discussion and conclusions

This paper presents the methodology for a learning and inference mechanism for design optimization problem formulation. The method is based on a statistical pattern extraction view of machine learning, where syntactic patterns of previous formulations are hypothesized to be the foundation for representing semantic design concepts, relationships and patterns. The main contribution of the method is in its ability to extract hidden and explicit design concepts, relationships and patterns expressed mathematically in the problem formulation, using a small number of training examples for a family of design problems. A possible contribution of the method lies in its potential application to large scale problems, where the design patterns to be extracted are not made apparent by simple human observation of the problem statement. We are currently in the process of applying and testing the method to large-scale problems, where the size of the problems is significantly larger in terms of the number of design variables, parameters and constraints, and the method is being used for symbolic problem model decomposition. It is a known fact that SVD calculation algorithms scale well. The main aim of the application to large scale problems is to test whether the method scales well for design optimization. The preliminary results appear promising and a full "design formulation assistance" system is being developed in MATLAB. With this tool, the designer will be able to state problems symbolically, define experiences and experience databases, and query the experience database for problem (re)-formulation advice.

References

1. Papalambros PY, Wilde DJ (2000) Principles of optimal design. Cambridge University Press
2. Nath G, Gero JS (2004) Learning while designing. AIEDAM 18 (4): 315-341
3. Gelsey A, Schwabacher M, Smith D (1998) Using modeling knowledge to guide design space search. Artificial Intelligence 101: 35-62
4. Suwa M, Gero JS, Purcell T (2000) Unexpected discoveries and s-inventions of design requirements: important vehicles for a design process. Design Studies 21(6): 539-567
5. Moss J, Cagan J, Kotovsky K (2004) Learning from design experience in an agent-based system. Research in Engineering Design 15: 77-92
6. Strang G (2003) Introduction to linear algebra. Wellesley-Cambridge Press
7. Kalman D (1996) A singularly valuable decomposition: the SVD of a matrix. The College Mathematics Journal 27 (1): 2-23
8. Balachandran M, Gero JS (1987) A knowledge based approach to mathematical design modeling and optimization. Engineering Optimization 12(2): 91-115
9. Mackenzie CA, Gero JS (1987) Learning design rules from decisions and performances. Artificial Intelligence in Engineering 2(1): 2-10
10. Schwabacher M, Ellman T, Hirsh H (1998) Learning to set up numerical optimizations of engineering designs. AIEDAM 12: 173-192
11. 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
12. Campbell MI, Cagan J, Kotovsky K (2003) The A-design approach to managing automated design synthesis. Research in Engineering Design 14: 12-24
13. Landauer T K, Dumais S T (1997) A solution to Plato's problem: The latent semantic snalysis theory of acquisition, induction and representation of knowledge. Psychological Review, 104(2): 211 – 240
14. Dong A, Agogino AM: (1997) Text analysis for constructing design representations. Artificial Intelligence in Engineering, 11 (2): 65-75
15. Hill A, Song S, Dong A, Agogino A (2001) Identifying shared understanding in design teams using document analysis. 13th International Conference on Design Theory and Methodology, DETC2001/ DTM-21713, ASME Press, Chicago, IL
16. Moss J, Kotovsky K, Cagan J (2004) Cognitive investigations into knowledge representation in engineering design. In JS Gero (ed), Design Computing and Cognition '04, Kluwer: 97-116
17. Michelena N, Agogino A (1988) Multiobjective hydraulic cylinder design, Journal of Mechanisms. Transmissions and Automation in Design 110: 81-87