# A DECOMPOSITION ALGORITHM FOR PARAMETRIC DESIGN

**J. M. Jauregui-Becker[1], W. O. Schotborgh[2]**
(1, 2, 3) University of Twente

## ABSTRACT
This paper presents a recursive division algorithm to decompose an under constraint parametric design problem. The algorithm defines the separation of the problem at the hand of two complexity measures that are calculated for each parameter in the problem, namely, the effort E and the influence Inf. The result from applying this algorithm is a sequence indicating the order in which parameters can be instantiated by, for example, a random value generator, or be calculated solving an equation. The characteristic of this algorithm is that it considers the structure of the equations in the problem to derive a solving strategy instead of considering its mathematic details. The advantage of doing so, from a design point of view, is that the algorithm is not limited to handling any specific types of equations (like for example pure algebraic or differential). In opposition, the algorithm is capable of handling different combinations of types of knowledge for determining a solving procedure.

*Keywords: Algorithm, decomposition method, parametric design, constraint solving.*

## 1 INTRODUCTION
Advances in technology are driving modern products towards further miniaturization, better quality, more functionality, and yet cheaper prices. To cope with this trend, engineers often redesign existing products to meet new requirements and optimize performances. When the type of components, variables and relations are known on forehand, and the design challenge lies in finding the right instances, the problem can be regarded as routine design [1]. Although routine design occurs within a well defined domain of knowledge, many problems demonstrate the complexity that they can exhibit [2]. In the case of parametric design, the complexity of the problem increases as the number of relations and parameter becomes larger. In order to deal with this complexity, great varieties of Constraint Solving (CS) and Optimization Techniques (OT) have been investigated and developed [3]. For design purposes, where the problems are under constrained, CS techniques are used to identify the space of solutions that satisfy the constraints [4]. Optimization techniques focus on the maximizing or minimizing a given objective function, in addition to the constraint satisfaction problem [5]. This paper deals with CS. Several approaches can be found on literature for solving CS problems. For example, one approach consists in using the derivatives of the relations to determine in which direction to steer the solving process [3]. Another approach -being the one this paper will further focus on- consists of decomposing the large sets of equations and constraints into a set of smaller ones based on the structure of the relations and then solving it. Thus, the classic divide and conquer strategy. This type of approaches is regarded as geometric constraint solving, or just decomposition method. The advantage of such an approach is that it is capable of handling different types of models (e.g. algebraic, logic, etc) at the same time, which reassembles the condition of the knowledge involved in design, as described in [6].
This paper presents a recursive division algorithm to decompose an under constraint system. The algorithm defines the separation of the problem at the hand of two complexity measures that can be calculated for each of the parameters in the problem, namely, the effort *E* and the influence *Inf*. The effort measures the number of parameters that have to be known in order to resolve a parameter with a certain equation. The influence of a parameter measures the number of parameters that can be solved if this parameter would be known. Furthermore, the algorithm is also based on four states a parameter can have: unknown, known, driver and driven. At the beginning of the problem, all parameters have the state unknown. Then, as the designers set requirements on the problem, some parameter take the state known. A driver state corresponds to parameters that are instantiated by an algorithm (e.g. randomly) or by a design decision, while a driven state defines a parameter that can be calculated in a

relation where the values of all other parameters are known. By assessing the values of the effort and the influence of each parameter, the algorithm chooses which parameter should become driver and which becomes driven as consequence of that decision. The algorithm can be used to determine problem partitions such that over constrained sub problems arise as solutions are generated as well as to suggest the sequence in which unknown parameter should be instantiated, and which equations are required for doing so.

The rest of this paper is structured as follows: Section 2 presents a brief description of related work and how this research relates to it. Section 3 presents Knowledge Graphs (KG) as a means of representing the parameters and relations contained in a parametric design problem. Here, it is also presented how to transform the KG into an adjacency matrix, which is further defined as the Knowledge Graph Matrix, or KGM. Section 4 introduces the concepts of effort and influence, and presents equations for its calculations. Section 5 defines the states a parameter can undergo, as well as possible state transitions. Section 6 presents the KGM transformation, which defines the changes that occur to the KGM each time a state transition occurs. Section 7 indicates how to identify driver and driven states. Section 8 introduces the algorithm, and, finally, Section 9 describes an example.

## 2 RELATED WORK

Four main types of decomposition approaches for constraint systems can be identified in literature [7]. One approach consists in identifying subsystems that are straight solvable. A second approach aims at identifying points of weakness in the constraint system for partitioning the problem. A third one consists in identifying subsystems as solvable provided that the complementary subsystem is solvable as well. The fourth, denominated recursive division methods, works by iteratively splitting the constraint system into components, themselves subject to further splitting. In order to meet real-life applicability, there are two desirable properties that such a method should have: generality and reliability. The algorithm presented in this paper is a recursive division algorithm characterized by the following aspects:

1. The directed graph consists of nodes denoting the parameters, and arcs denoting its relations, while the adjacency graph of the methods in [7] is composed of nodes that represent both the parameters and the relations. Arcs are used to describe which parameters are involved in which relations. The adjacency matrix of this graph allows solving asymmetric problems (those in which at least one relation contains a parameter that is not solvable as function of the others).
2. The decomposition is based on two complexity measurements, namely, the effort and the influence. Because of the generality of the approach, the algorithm can be used in different types of CS problems, as for example, geometric and topological problems.

A proper benchmarking is required to objectively identify the differences between the method presented in this paper and other described in literature.

## 3 KNOWLEDGE GRAPH MATRIX (KGM)

The first step required for using the algorithm described in this paper, is to gather the relations involved in the parametric design problem and construct the Knowledge Graph Matrix. This section explains the rationales for assembling such a matrix.

### 3.1 Knowledge Graph

Design problem knowledge can explicitly be modeled as function of parameters and three types of rules (resolve, constrain and expand rules) as it is demonstrated in [8]. Resolve rules determine the value of a parameter, such as an equation, logic, or random guess. Constrain rules check the validity of a (partial) solution, which can be an inequality or also a logic reasoning action. Expand rules govern the expansion of designs with multiple topological elements. This knowledge is explicitly written as mathematical model, but can also be graphically presented using knowledge graphs.

Knowledge Graphs represent the network of deterministic knowledge and possible paths through the network to resolve all parameters and find a solution. Knowledge graphs (KG) consist of nodes and directed edges. Nodes represent parameters and the directed edges describe the relations among each parameter. The directions of the edges are pointed toward the parameter to resolve. For example, if node D has N ingoing edges, the parameter represented by node D requires N other parameters to be known in order to resolve it. The knowledge graphs edges are labelled according to the relations to whom they belong.

An example of such a knowledge graph is shown in Figure 1. This graph represents equation (1) and equation (2), used in the design of a compression spring. The first equation describes the winding ratio $w$ as a fraction of the spring diameter $D$ and the wire diameter $d_{wire}$. The second describes the maximum spring compression $L_c$ as function of the number of coils n and the wire diameter $d_{wire}$.

$$w = \frac{D}{d_{wire}} \tag{1}$$

$$L_c = n \cdot d_{wire} \tag{2}$$

Node $w$ has two edges directed toward it, meaning that its value can be resolved if the values of $D$ and $d_{wire}$ are known. As shown in Figure 1, the parameter value of $d_{wire}$ has no edges toward it. This means $d_{wire}$ cannot be resolved with this equation. The reason is that the allowed values of $d_{wire}$ are discontinuous (specified by a DIN standard), and a calculated value from equation (1) is not guaranteed to be exactly a DIN value. Therefore, $d_{wire}$ has to be resolved by another rule before equation (1) can be used to resolve either $w$ or $L_c$.

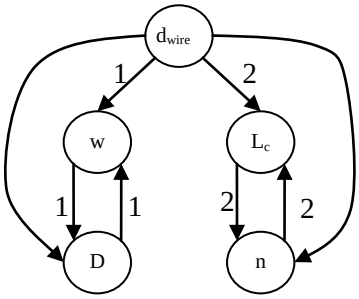


*Figure 1: Knowledge graph of equation (1) and equation (2).*

### 3.2 Knowledge Graph Matrix Assemble

The adjacency matrix of the knowledge graph is regarded here as the Knowledge Graph Matrix. In general, an adjacency matrix of a directed **G** on *n* vertices is the $n \times n$ matrix where the non diagonal entry $a_{ij}$ is the number of edges from vertex *i* to vertex *j*, and the diagonal entry $a_{ii}$ is number of loops at vertex. However, given the form of knowledge graphs (KG), no loops are present and therefore the diagonal entries remain empty. Furthermore, each entry in the matrix is accompanied by the equation *id* related to that entry. For the case of the knowledge graph in Figure 1, the corresponding adjacency matrix would take the following form:

$$KGM = \begin{pmatrix} & d_{wire} & D & w & L_c & n \\ d_{wire} & 0 & 1(1) & 1(1) & 1(2) & 1(2) \\ D & 0 & 0 & 1(1) & 0 & 0 \\ w & 0 & 1(1) & 0 & 0 & 0 \\ L_c & 0 & 0 & 0 & 0 & 1(2) \\ n & 0 & 0 & 0 & 1(2) & 0 \end{pmatrix} \tag{3}$$

## 4 EFFORT AND INFLUENCE

Two measures are used to assess how each node of the knowledge graph is related to the others. These two measures are used to determine which parameters are "easier" to solve and which will help solving most of the left unknown parameters. The first case corresponds to the parameter with the lowest effort, while the second corresponds to those with the largest influence.

#### *Effort*

Each node, or parameter, has an effort (*E*) for each of the relations in which it is included. The effort represents the number of parameters that have to be known in order to resolve a parameter with a

certain equation. In a knowledge graph, the effort in one relation of one parameter corresponds to the sum of all the edges pointing towards that node. Using the matrix representation, the effort of a parameter *p* in a relation *n* is calculated by:

$$E_{p,n} = \sum_{i=1}^{l} KGM(i, p) \tag{4}$$

where *l* equals the number of rows in the KGM. Furthermore, if one column of the matrix has entries that relate this parameter to others, the effort of this parameter is set to 1 to represent the effort the parameter requires for solving itself.

***Influence***

Each node, or parameter, has an influence (*I*) that measures the number of relations in which this parameter is present. This equals the sum of all relations present in the row corresponding to a given parameter and is calculated as follows:

$$Inf_p = \sum_{i=1}^{l} KGM(p, i) \tag{5}$$

where *l* equals the number of columns of the KGM. If one row does not contain entries, the parameter related to it has an influence equal to zero. This means that this parameter cannot be used to solve others.

*Table 1: Efforts and influences at problem class.*

| Parameter | $d_{wire}$ | *D* | *w* | *n* | *Lo* |
|---|---|---|---|---|---|
| Effort | 1 | 2(1) | 2(1) | 2(2) | 2(2) |
| Influence | 4 | 1 | 1 | 1 | 1 |

In Table 1 the efforts and influences of the KGM shown in equation (3) are presented. As the table shows, the parameters *D, w, n* and *Lc* have all effort equal to $2$ and influence equal to $1$, while the parameter $d_{wire}$ has an effort equal to $1$ (which is the effort of solving itself) and an influence equal to $4$. This indicates that for this case $d_{wire}$ is the easiest variable to solve as well as it is the parameter that if solved will help the most parameters to be solved.

## 5 PARAMETER STATES

Parametric design problems are formulated at the hand of known and unknown parameters related by constraints and equalities. Known parameters impose requirements on the problem, while unknown parameters are the ones the designer is interested in solving. Therefore, the two types of states a parameter can have when formulating a design problem is ***known*** and ***unknown***. Unknown parameters become known according to two possibilities: (1) as a design decision of the designer or, in the case of automated design, by an algorithm (e.g. randomly); or (2) by calculating it with a relation where precisely one parameter is unknown. In this method, these cases are regarded as two extra states possibilities of a parameter. The first one is termed as ***driver*** state, as the parameter being instantiated will drive the calculation of other unknown parameter. The second case is termed ***driven*** state, as the instantiation is driven by the previous algorithmic instantiation of other parameters. Thus, the four states a parameter can have in this approach are: known, unknown, driver and driven. In these terms, the goal of solving a design problem is to transform parameters with unknown states into parameters with known, driver and driven states such that valid solutions are found or an objective function is maximized or minimized.

The three possible types of state transition are:

1. ***Unknown → known:*** occurs when the problem statement is being made by attributing a value to one parameter as a requirement the design has to satisfy.
2. ***Unknown → driver:*** occurs by attributing a value to a parameter according to a design decision or an algorithmic step (e.g. by randomly attributing a value).
3. ***Unknown → driven:*** occurs when the effort of a parameter becomes zero after KGM recalculation. This means that this variable can be calculated by using the relation attributed to the effort that became zero.

## 6 KGM TRANSFORMATIONS

As it will be explained later, each time a state transition occurs, the KGM changes and has to be recalculated. Recalculating the KGM is performed by:

1. Eliminating all entries in the row and the column that corresponds to that parameter from the KGM.
2. Recalculating all efforts and influences according to equations (4) and (5).

Consider the example of the KGM in matrix (3). If one assumes that the parameter *D* (spring diameter) is set as known and all the others are unknown, the KGM is transformed as shown in equation (6), obtaining the efforts and influences shown in Table 2.

$$KGM = \begin{pmatrix} & d_{wire} & D & w & L_c & n \\ d_{wire} & 0 & 0 & 1(1) & 1(2) & 1(2) \\ D & 0 & 0 & 0 & 0 & 0 \\ w & 0 & 0 & 0 & 0 & 0 \\ L_c & 0 & 0 & 0 & 0 & 1(2) \\ n & 0 & 0 & 0 & 1(2) & 0 \end{pmatrix} \tag{6}$$

## 7 IDENTIFYING DRIVER AND DRIVEN

In essence, the KGM algorithm consists on choosing drivers and identifying which parameters become driven as consequence of that decision. Drivers are identified by choosing the parameter with the lowest effort. If the value of the minimum effort is shared by more than one parameter, then the parameter with the highest influence among them is chosen. By doing so, it is avoided that two parameters become over-constrained as the algorithm progresses. This also minimizes the effort of other parameters. In the case that the minimum effort and the highest influence are shared by more than one parameter, either one of the parameters can be chosen as driver parameter.

Consider the efforts and influences in Table 1. As it can be seen, the parameter $d_{wire}$ has the lowest effort. As no other parameter has an effort equal to that of $d_{wire}$, this parameter becomes automatically a driver parameter. Drivens are identified as those parameters whose effort becomes 0 (zero) after a KGM transformation has taken place. In this case, this parameter can be calculated by using the relation corresponding to the effort that became 0 (zero).

## 8 THE ALGORITHM

Having presented the KGM, its transformations and a method for identifying drivers and drivens, the algorithm for recognizing the sequence in which to solve a parametric design problem is presented in Figure 2.

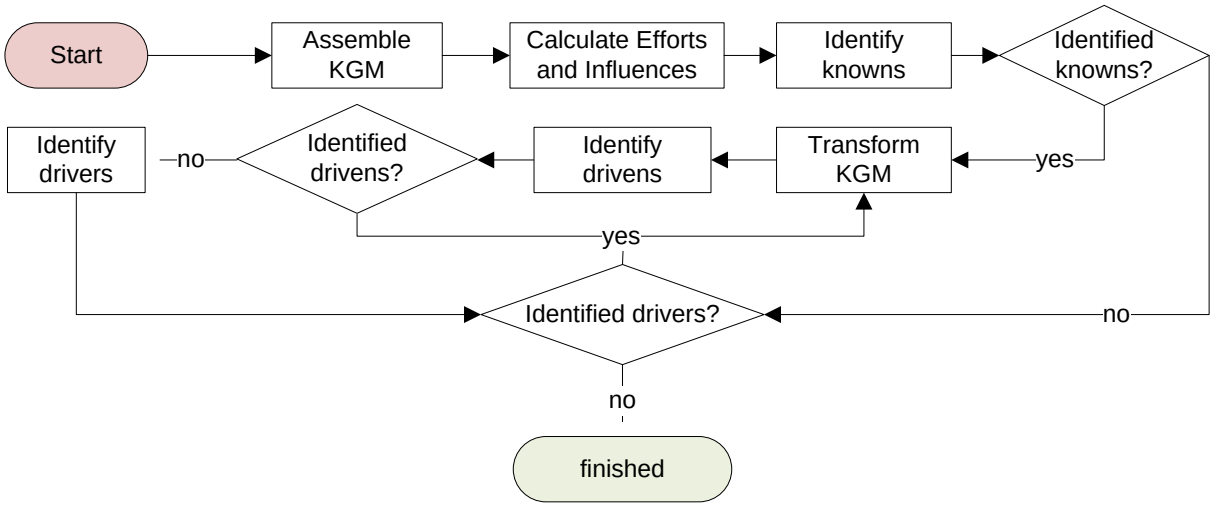


*Figure 2: KGM solving algorithm.*

Table 2 shows the results of applying the algorithm on the example shown in Table 1. As the table shows, the state transition occurs when setting the value of $D$ to known in step 1. In step 2, the recalculated efforts and influences are shown. Here, the variable $d_{wire}$ is identified as the best to be changed into driver (*). It can also be seen that the variable $w$ gets an influence of zero (**), as this parameter cannot influence solving others. In step 3, the recalculated KGM measurements indicate that w becomes driven, as its effort drops to zero (***). As $w$ is only contained in relation 1, its effort only depends on this equation. Therefore, the value of $w$ can be calculated by using equation 1 in combination with the previously defined values of $D$ and $d_{wire}$. In step 2, the parameter $n$ is chosen as driver, as no other parameter became driven. However, parameter $L_o$ could have also been chosen, as its effort and influence values equals the one of $n$. As result of a new KGM transformation, parameter $L_o$ becomes driven with equation (2).

*Table 2: Result of KGM transformations in example.*

| Parameter | | $d_{wire}$ | $D$ | $w$ | $n$ | $L_o$ |
|---|---|---|---|---|---|---|
| Step 1 | Effort | 1 | 2(1)* | 2(1) | 2(2) | 2(2) |
| | Influence | 4 | 1 | 1 | 1 | 1 |
| Step 2 | Effort | 1** | 0 | 1(1) | 2(2) | 2(2) |
| | Influence | 3 | 0 | 0 | 1 | 1 |
| Step 3 | Effort | 0 | 0 | 0*** | 1(2) | 1(2) |
| | Influence | 0 | 0 | 0 | 1 | 1 |
| Step 4 | Effort | 0 | 0 | 0 | 1(2)** | 1(2) |
| | Influence | 0 | 0 | 0 | 1 | 1 |
| Step 5 | Effort | 0 | 0 | 0 | 0 | 0*** |
| | Influence | 0 | 0 | 0 | 0 | 0 |

*State transition: * unknown-known, ** unknown-driver, *** unknown-driven*

The result of applying the algorithm is a sequence of proposed parameters and state transitions. The sequence can be outlined by keeping track on the order in which parameter states change from unknown to knowns, drivers or drivens. In the case of a driven parameter, the relations through which they are calculated have to be recorded too. After this algorithm is applied to a given parametric design problem, the resulting sequence can be used by a constraint solving or optimization algorithm to search for the numerical solution of the problem. For the example in Table 2, the resulting sequence is shown in Figure 3.

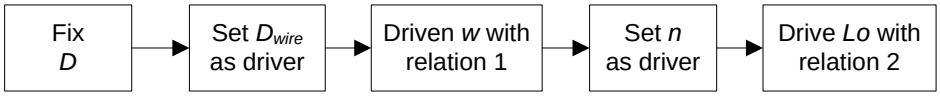


*Figure 3: Example of strategy.*

Although the main goal of this algorithm is to define the sequence in which an under constraint parametric design problem could be solved, it is also successful in identifying well-constrains and over-constrains situations. Well-constrained, or systems of equations, are identified when the effort of more than one parameter becomes zero in more than one relation at the same time. On the other hand, over-constrained problems are identified when more than one effort of one parameter drop to zero at the same time. The interpretation of the last is that this parameter can be calculated by more than one relation, and therefore is over-constrained.

## 9 EXAMPLE: COMPRESSION SPRING

The parametric design of a compression spring is used to demonstrate the algorithm at the hand of one combination of known and unknown parameters. A compression spring is modeled as a single element with 11 parameters and 6 relations (material is kept constant) [9]. The parameters are stated in Table 3 and the relations in equation (7) - (12). An ID has been attributed to each relation. Figure 4 shows the KG of this example with the IDs of each relation appearing along the arcs. At the hand of the KG, the KGM is assembled and the efforts and influences of each parameter are calculated, as shown in Table 4 and 5 respectively.

*Table 3: Parameters considered in compression spring design.*

| **Parameter** | **Description** | **Parameter** | **Description** |
|---|---|---|---|
| D | mean diameter | $L_c$ | maximum compressed length |
| d | wire diameter | s | compression |
| n | number of coils | sc | maximum compression |
| A | space between coils | R | spring constant (stiffness) |
| $L_0$ | uncompressed length | F | force at compression |
| $L_s$ | compressed length | | |

$$R = \frac{G}{8} \cdot \frac{d^4}{D^3 \cdot n} \mapsto ID = 1 \quad (7)$$

$$F = R \cdot s \mapsto ID = 2 \quad (8)$$

$$L_0 = s + L_s \mapsto ID = 3 \quad (9)$$

$$L_0 = sc + L_c \mapsto ID = 4 \quad (10)$$

$$L_c = n \cdot d \mapsto ID = 5 \quad (11)$$

$$L_0 = n \cdot (A + d) \mapsto ID = 6 \quad (12)$$

*Table 4: KGM of compression spring design*

| | D | d | n | A | Lo | Ls | Lc | s | sc | R | F |
|---|---|---|---|---|---|---|---|---|---|---|---|
| D | | 1(1) | 1(1) | | | | | | | 1(1) | |
| d | 1(1) | | 1(1)<br>1(5)<br>1(6) | 1(6) | 1(6) | | 1(5) | | | 1(1) | |
| n | 1(1) | 1(1)<br>1(5)<br>1(6) | | 1(6) | 1(6) | | 1(5) | | | 1(1) | |
| A | | 1(6) | 1(6) | | 1(6) | | | | | | |
| Lo | | 1(6) | 1(6) | 1(6) | | 1(3) | 1(4) | 1(3) | 1(4) | | |
| Ls | | | | | 1(3) | | | 1(3) | | | |
| Lc | | 1(5) | 1(5) | | 1(4) | | | | 1(4) | | |
| S | | | | | 1(3) | 1(3) | | | | 1(2) | 1(2) |
| Sc | | | | | 1(4) | | 1(4) | | | | |
| R | 1(1) | 1(1) | 1(1) | | | | | 1(2) | | | 1(2) |
| F | | | | | | | | 1(2) | | 1(2) | |

*Table 5: Initial efforts and influences.*

| Parameters | | D | d | n | A | Lo | Ls | Lc | s | sc | R | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Start | E | 3(1) | 3(1)<br>3(6)<br>2(5) | 3(1)<br>3(6)<br>2(5) | 3(6) | 3(6)<br>2(4)<br>2(3) | 2(3) | 2(5)<br>2(4) | 2(2)<br>2(3) | 2(4) | 2(2)<br>3(1) | 2(2) |
| | Inf | 3 | 8 | 8 | 3 | 7 | 2 | 4 | 4 | 2 | 5 | 2 |

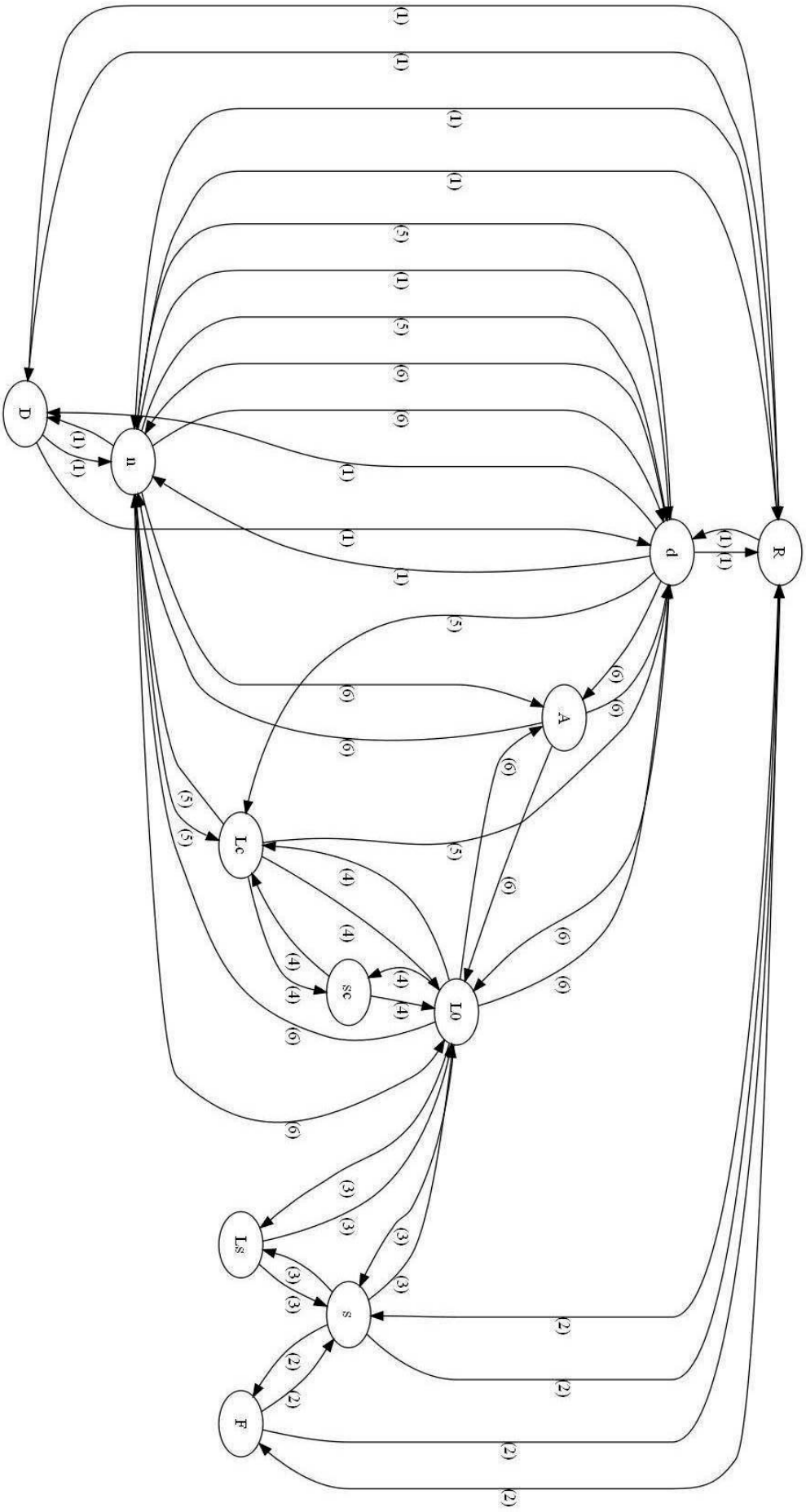


*Figure 4: Knowledge graph of spring design.*

*Table 6: Problem instance of spring design example.*

| Parameter | State | Parameter | State | Parameter | State |
|---|---|---|---|---|---|
| D | unknown | L0 | Known | sc | unknown |
| d | unknown | Ls | unknown | R | unknown |
| n | unknown | Lc | Known | F | Known |
| A | unknown | s | unknown | | |

*Table 7: Results of KGM transformation in example.*

| Parameters | | D | d | n | A | Lo | Ls | Lc | s | sc | R | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Step 1 | E | 3(1) | 3(1)<br>2(6)<br>1(5) | 3(1)<br>2(6)<br>1(5)<br>** | 2(6) | 0<br>* | 1(3) | 0<br>* | 2(2)<br>1(3) | 0(4) | 2(2)<br>3(1) | 0<br>* |
| | Inf | 3 | 6 | 6 | 2 | 0 | 1 | 0 | 2 | 0 | 4 | 0 |
| Step 2 | E | 2(1) | 2(1)<br>1(6)<br>0(5)<br>*** | 0 | 1(6) | 0 | 1(3) | 0 | 1(2)<br>1(3) | 0 | 1(2)<br>2(1) | 0 |
| | Inf | 2 | 3 | 0 | 1 | | 1 | 0 | 2 | 0 | 3 | 0 |
| Step 3 | E | 1(1) | 0 | 0 | 0(6)<br>*** | 0 | 1(3) | 0 | 1(2)<br>1(3) | 0 | 1(2)<br>1(1) | 0 |
| | Inf | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 2 | 0 | 2 | 0 |
| Step 4 | E | 1(1) | 0 | 0 | 0 | 0 | 1(3) | 0 | 1(2)<br>1(3)<br>* | 0 | 1(2)<br>1(1) | 0 |
| | Inf | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 2 | 0 | 2 | 0 |
| Step 5 | E | 1(1) | 0 | 0 | 0 | 0 | 0(3)<br>*** | 0 | 0 | 0 | 0(2)<br>1(1)<br>*** | 0 |
| | Inf | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Step 6 | E | 0(1)<br>*** | 0 | 0 | 0 | 0 | 0(3)<br>*** | 0 | 0 | 0 | 0(2)<br>1(1)<br>*** | 0 |
| | Inf | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*State transition: * unknown-known, ** unknown-driver, *** unknown-driven*

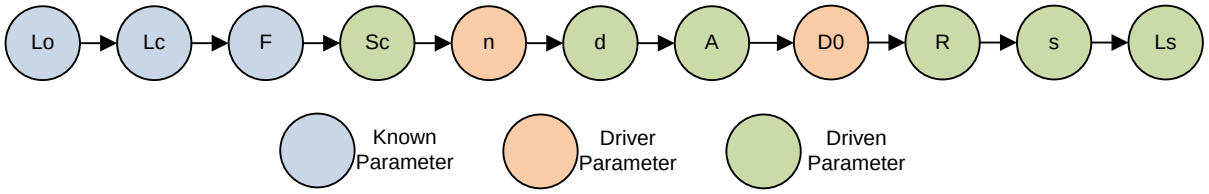


*Figure 5: Instantiating order of parameters in spring example.*

The combinations of known and unknown that have been used to demonstrate the algorithm is shown in Table 6. In Table 7 the efforts and influences that result from applying the algorithm are presented, while Figure 5 shows the resulting sequence. As it can be seen, some parameters have more than one effort value attributed. This is the consequence of having some parameters present in more than one equation. The table also shows that the parameter sc becomes driven with equation (10) after recalculating the KGM for the known parameters. This is due to the fact that all parameters in relation (10), except sc, have an attributed value. In step 4, it can be seen that although the parameters s, R and

Ls have the same minimum value of the effort (effort = 1), the parameter s is chosen as driver. The reason for doing so is that the influence of s is larger than that of the parameters R and Lo, which means that attributing a value to this parameter will reduce the effort of two other parameters, speeding the course of the algorithm.

## 10 SUMMARY

This paper presented a recursive division algorithm to decompose under-constraint parametric design problem. The algorithm determines the order in which parameters can be instantiated in an under-constrained parametric design problem. The algorithm gets its name from the Knowledge Graph Matrix used for representing the structure of parameters and relations. The algorithm is based on two complexity measures that can be calculated for each parameter, namely, the effort and the influence. The effort relates to the number of parameters that have to be known in order to solve another parameter. The influence is related to the number of parameters that can be solved if a given parameter is known. The algorithm finds a solving order by assessing which parameters have low efforts and high influences. Results from applying the algorithm show that it is successful in determining sequence for solving parameters in an under-constrained problem.


## ACKNOWLEDGEMENTS

The authors gratefully acknowledge the support of the Dutch Innovation Oriented Research Program 'Integrated Product Creation and Realization (IOP-IPCR)' of the Dutch Ministry of Economic Affairs.


## REFERENCES


[1] Gero, J.S. and Kannengiesser, U., The situated function-behaviour-structure framework. *Design Studies*, 2004, 25, pp373-391.

[2] Ameri, F., Summers, J., Mocko, G. and Porter, M., Engineering design complexity: an investigation of methods and measures. *Research in Engineering Design*, 2008, 19(2), pp161-179.

[3] Hooker, J.N., *Integrated Methods for Optimization*. (Springer, 2007).

[4] Kumar, V., Algorithms for constraint satisfaction problems: a survey. *AI magazine*, 1992, 13(1), pp32-44.

[5] Papalambros, P.Y. and Wilde, D.J., *Principles of Optimal Design- Modeling and Computation*. (Cambridge University press, 2000).

[6] Schotborgh, W.O., Kokkeler, F.G.M., Tragter, H., Bommhoff, M.J. and van Houten, F., A Generic Synthesis Algorithm for Well-Defined Parametric Design. *Proceedings of the 18th CIRP Design Conference*, 2008.

[7] Jermann, C. and Trombettoni, G., Decomposition of geometric constraint systems: a survey. *IJCGA*, 2006, 16.

[8] Schotborgh, W.O., *Knowledge engineering for design automation*. (University of Twente, 2009).

[9] Norton, R.L., *An Introduction to the Synthesis and Analysis of Mechanisms and Machines*. (McGraw-Hill College, 2003).



Contact: Juan M. Jauregui-Becker
University of Twente
Faculty of Engineering Technology
Laboratory of Design, Production and Management
P.O. Box 217, 7500 AE Enschede
The Netherlands
T: +31 53 489 4025
F: +31 53 489 3631



Juan Jauregui-Becker currently works as assistant professor at the University of Twente, in The Netherlands. His PhD thesis investigated the development of a computational design synthesis method. He received his BSc degree in mechanical engineering at the University of Los Andes in Venezuela and his master degree in mechanical automation at the University of Twente in The Netherlands.