Research in Engineering Design (1996) 2: 116–124
© 1996 Springer-Verlag London Limited

An Approach to Functional Synthesis of Solutions in Mechanical Conceptual Design. Part III: Spatial Configuration

A. Chakrabarti and T. P. Bligh
The Engineering Design Centre, Department of Engineering, University of Cambridge, Cambridge, UK

Abstract. *These three papers describe an approach to the synthesis of solutions to a class of mechanical design problems; these involve transmission and transformation of mechanical forces and motion, and can be described by a set of inputs and outputs. The approach involves (1) identifying a set of primary functional elements and rules of combining them, and (2) developing appropriate representations and reasoning procedures for synthesizing solution concepts using these elements and their combination rules; these synthesis procedures can produce an exhaustive set of solution concepts, in terms of their topological as well as spatial configurations, to a given design problem.*

This paper (Part III) describes a constraint propagation procedure which, using a knowledge base of spatial information about a set of primary functional elements, can produce possible spatial configurations of solution concepts generated in Part II.

Keywords. Computer aided design; Conceptual design; Concept generation; Constraint propagation; Functional modelling; Functional reasoning; Functional synthesis; Mechanical design; Spatial configuration; Spatial reasoning; Transmission design

1. Introduction

In Part II (Kind Synthesis, see Sections 2, 4 and 6), procedures for kind synthesis are described which, using the kind transformations of the available structures in a knowledge base (created using the representation constructed described in Section 7, Part I), can produce graph-structures which solve instantaneous kind-requirements of a transmission design problem. Any of the solution concepts so produced must also satisfy the orientation requirements of the problem, i.e., the inputs and outputs should have specified orientations in space. Each solution concept which satisfies the orientation requirements must, in turn, satisfy the sense requirements of these orientations. The solutions so produced should have the possibility of satisfying the position and magnitude requirements of the inputs and outputs.

The following sections elaborate procedures developed for producing concept-configurations which can satisfy the orientation and sense requirements of a problem. Indicated also is how the position and magnitude requirements of the problem could be used as constraints for checking the solutions for validity in the more detailed design phases.

Correspondence and offprint requests to: Dr A. Chakrabarti, Department of Engineering, University of Cambridge, Trumpington Street, Cambridge CB2 1P2, UK.

2. Orientation Configuration

Even within the "orthogonality restrictions" (see Section 5.2, Part I), it might be possible to orient a solution concept, produced using the kind synthesis procedure (Part II), in more than one way in space. The problem of orientation configuration, within the confines of the *orthogonality restrictions*, to to produce the valid orientations, if any, of the solution concepts synthesized by the kind synthesis procedure.

This could be viewed as a *constraint propagation* problem. The solution, in the most general case, could be regarded as a *network* having a set of *nodes* (I/O points) and a set of *arcs* (structures joining the *nodes*). The constraints are given by the specific orientation values ($i/j/k$) that some of the nodes (in this case the I/O nodes) must have. As the orientation transformations for each of the arcs (i.e., structures) are known, the orientation configuration problem for a given solution is to enumerate all the possible distinct orientations of the solution (i.e., its *arcs* and *nodes*) which are compatible with the specified orientation constraints.

To demonstrate how the constraint propagation

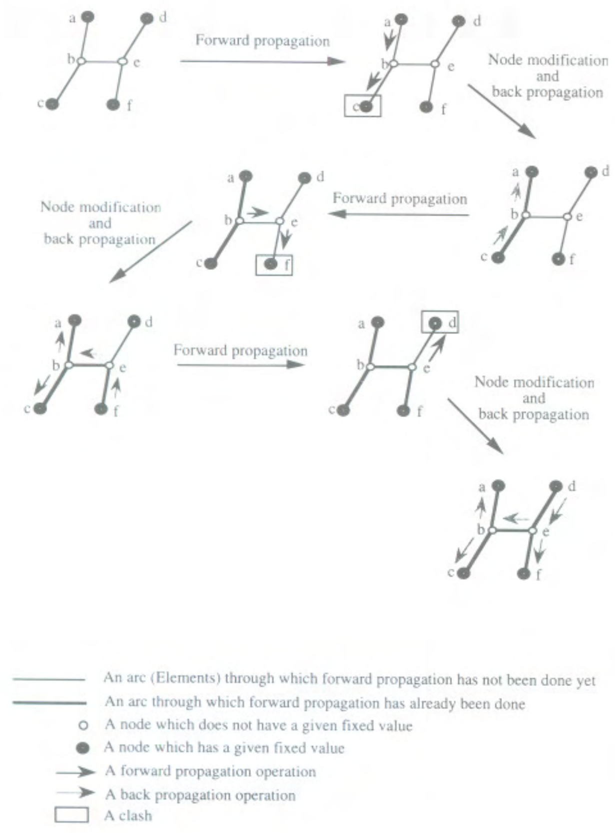

Forward propagation
Node modification and back propagation
Forward propagation
Node modification and back propagation
Forward propagation
Node modification and back propagation
a
b
c
d
e
f
An arc (Elements) through which forward propagation has not been done yet
An arc through which forward propagation has already been done
A node which does not have a given fixed value
A node which has a given fixed value
A forward propagation operation
A back propagation operation
A clash

Fig. 1. The Constraint propagation procedure.

procedure works (for its pseudo-code, see Appendix A), take the example network in Fig. 1. The network has six nodes, and five arcs connecting them. Among these nodes, four nodes (a, c, d, f) have given (orientation) values. Starting from any of these nodes, and using the spatial relationships between the nodes, the (orientation) values of the neighbouring nodes are calculated. This is called *forward propagation*. In this case, if propagation starts from node a, one possible route for forward propagation could be a-to-b-to-c. Node c, however, already has a given value. Thus a *clash*, as it is called here, occurs. The intersection between the new and the existing values of the node is taken as its modified value, and this operation is called *node modification*. Now, the effect of this modification is propagated back through that part of the network which is connected to this node, and through which forward propagation has already been done. This operation is called *back propagation*, and in the example case, this would modify the values from c-to-b-to-a. Forward propagation can now start for some un-propagated part of the network, for instance from b-to-e-to-f. At node f, a clash occurs again. The node-value is modified, and its effects back-propagated though f-to-e-to-b-to-a-and-c. Forward propagation

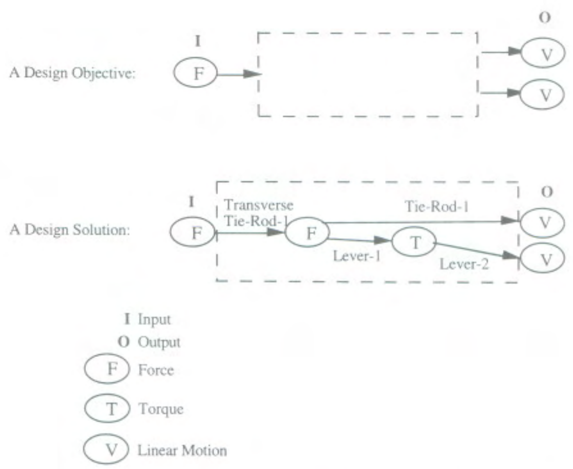


Fig. 2. A kind synthesis example: the toilet-door lock problem and one of its SIMO Solutions..

can start again now, from e-to-d. A clash is flagged at node d, its value modified, and its effect is back-propagated through the whole network. At this point, the constraint propagation is complete, and the network contains consistent sets of values for its nodes; each such set is one possible (orientation) configuration of the network.

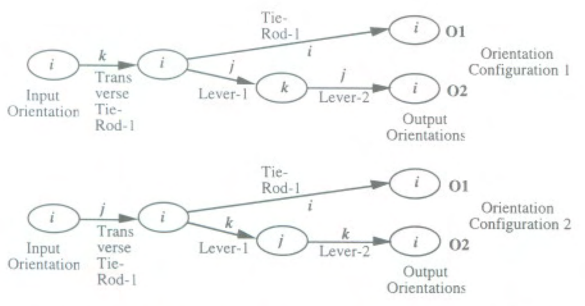
To show what the results look like, we take the orientation requirements of the *toilet-door lock* problem discussed in the SIMO kind synthesis example (Fig. 4 of Part II, reproduced here as Fig. 2), and try to configure the orientations (if any) for the solution illustrated in this figure. Here is the problem:

Orientation Transformation:

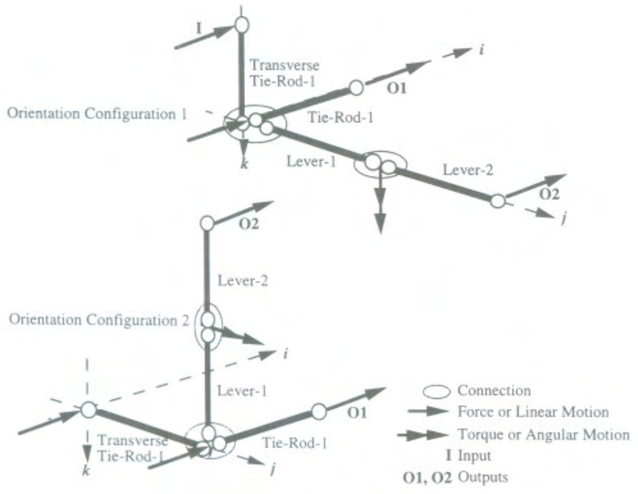
Input orientation $i \rightarrow$ Orientation of output–1i,
Orientation of output–2i,

From the available orientation transformations of the structures constituting the concept, the transformations compatible with the specified input and output orientations are first chosen. Using them, consistent orientation configurations of the intermediate structures are found.

The output of this procedure, for the above solution concept, would be a list of a set of ordered sequences of the I-, L- and O-vector orientations of the structures constituting the concept, which could be used for producing directed graphs or sketches of the orientations of the concept. The solution in Fig. 2, for instance, can be oriented in four different ways, two of which are represented, using the above information generated by the orientation configuration procedure, in the graph and the graphical representations shown in Figs 3a and b respectively.

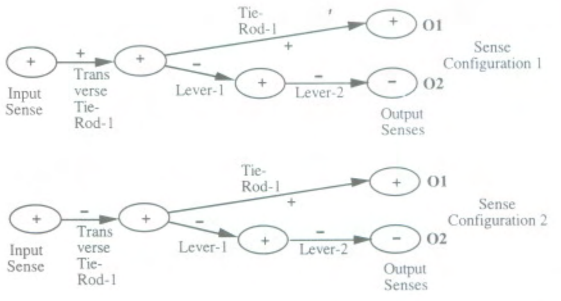


a. Graph Representation of Two Orientation Configurations of the Solution in Fig. 2

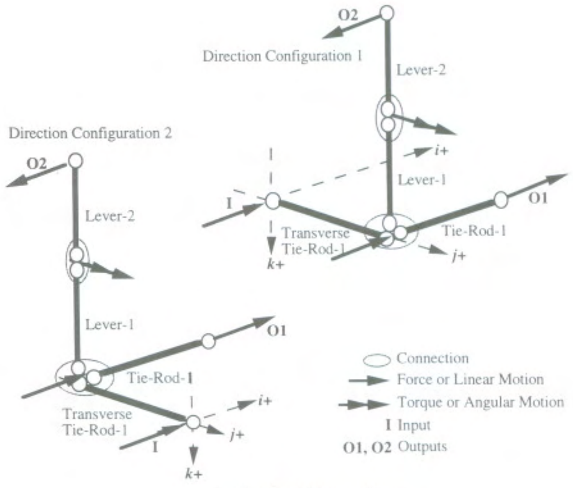


b. Graphical Representations

Fig. 3. Representation of some of the orientation configurations of the solutions in Fig. 2.



a. Graph Representation of Two Sense Configurations of Orientation Configuration 2 in Fig. 3



b. Graphical Representations

Fig. 4. Various representations of some of the direction configurations of the the orientation configuration 2 in Fig. 3.

3. Sense Configuration

While configurating the orientations of a solution concept, we deliberately avoided bothering about the senses in which the *I*-, *L*- and *O*-vectors should be directed. However, for each possible orientation configuration of a concept, there can be a number of sense configurations, as each vector can have a positive or a negative sense for the same orientation.

Sense Configuration can thus be framed as the problem of devising a procedure which would produce all possible sets of senses which a given orientation configuration of a given solution concept could have.

Considering a given orientation of a solution concept as a given *network* containing a set of *nodes* (i.e., I/O points) and *arcs* (i.e., constituting structures having known sense transformations), and considering the required sense-values (positive or negative) of the I/O nodes of the problem as the sense constraints, we can view sense configuration as a constraint propagation procedure which produces compatible sets of sense-values of the concept (i.e., its *nodes* and *arcs*).

As an illustrative example, the sense configuration procedure is applied here, for the purpose of transforming a positive input to a positive output-1 and a negative or positive output-2, on the orientation configuration 2 of Fig. 3. The procedure is similar to the orientation configuration procedure. The number of direction configurations (which consists of orientation and sense information) produced is 16, two of which are shown using the graph and the graphical representations in Figs 4a and b. The direction configuration 2 in Fig. 4 of the solution in Fig. 2 can be taken as an abstract representation of the spatial arrangement of a solution which is used in many existing *toilet-door lock* applications (see Fig. 7 for a schematic diagram of this solution).

4. Magnitude Constraint

During the conceptual design phase, one cannot ensure that a solution concept will satisfy a specified magnitude requirement (i.e., that the *I*- and the

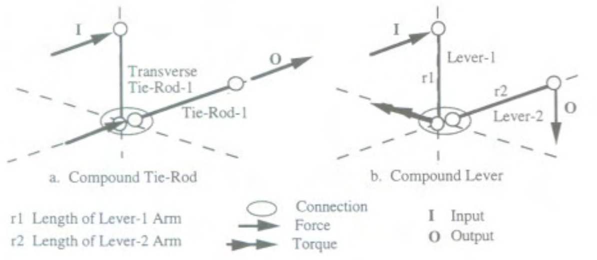


Fig. 5. In rare instances the magnitude constraint can be used to check the validity of a solution in the conceptual design phase itself.

O-vectors of the intended system should have specified magnitudes). When the magnitudes of the I- and the O-vectors of a structure are related by the physical attributes of the structure, as they usually are, the physical attributes can be adjusted to tune the magnitude transformation to fit the specific magnitude requirements. For instance, a lever, ideally, can be made to transform a force of any magnitude to a torque of any magnitude by adjusting its arm length.

However, it is possible to use the magnitude requirement as a constraint and express it in terms of the magnitude transformation factors of the constituent structures of a candidate solution. The magnitude transformation factor between the input and output torques of a compound lever (a serial combination of a lever and its inverse), for instance, can be calculated by multiplying the magnitude transformation factors of the two constituent levers, the values of which are usually undecided during the conceptual design phase. This constraint can be used to derive or validate the physical descriptions of solution concepts in the latter and better informed design phases.

In exceptional circumstances, however, this constraint can be used, during the conceptual design phase itself, to check the validity of a candidate solution concept. The solution shown in Fig. 5a, for instance, would not be able to amplify the magnitude of torque output (as the magnitude transformation factor is 1), whereas the solution of Fig. 5b can indeed be used for such a purpose (magnitude transformation factor is r_2/r_1, which can be adjusted).

5. Position Constraint

In some design problems, it may be required that the outputs of the desired solution be available at specific positions in space, while the input be supplied from a given position. Consequently, we need to know, at the earliest possible phase in design, whether a candidate solution could satisfy given position requirements. If the magnitudes of the L-vectors of the constituent structures of the candidate solution were known, the procedure to check the above would be to verify that the right side of each of the following vector equations, which gives the position changes achieved by a given solution, matched the left side, which expresses the position changes required by the problem where n inputs are transformed into m outputs:

For $m \leq n$:

$$\text{Position (system-output}_j) - \text{Position (system-input}_p) = \sum_{i=1}^{k_{jp}} L_i$$

where $j = 1, \ldots, m$; p stands for any system-input that belongs to the same connected-network as the system-output$_j$ in the above equation, and, the system-input$_p$ and the system-output$_j$ are directly or indirectly connected; L_i is the L-vector of the ith structure of the branch, of a concept, which connects the pth system-input to the jth output, and contains k_{jp} structures.

In this case, the set of all system-input$_p$s used in forming the m equations is equal to the complete set of system-inputs.

For $n \leq m$:

$$\text{Position (system-output}_p) - \text{Position (system-input}_j) = \sum_{i=1}^{k_{jp}} L_i$$

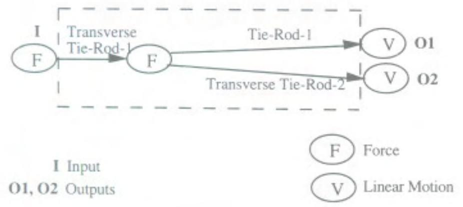
where $j = 1, \ldots, n$; p stands for any system output that belongs to the same connected-network as, and have a direct or indirect connection to, the system-input$_j$ in the above equation; L_i is the L-vector of the ith element of the branch, of a concept, which connects the pth system-input to the jth output, and contains k_{jp} structures.

In this case, the set of all system-output$_p$s used in forming the n equations is equal to the complete set of system-outputs.

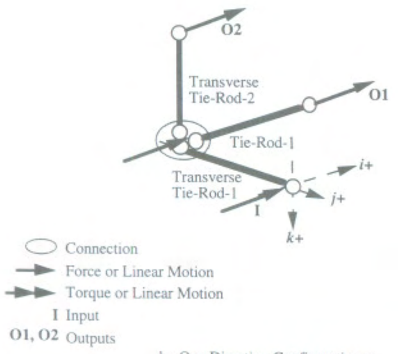
However, during the conceptual design, usually the magnitude of the L-vectors are not known, although it is possible to check this, even in conceptual design, in rare instances. Therefore, these equations can be kept as constraints to be checked during the latter phases of design.

6. Implementation, Validation and Performance

All the procedures were initially implemented on a Symbolics machine using the Common-LISP platform of the ART™ (Clayton, 1985) package; they presently

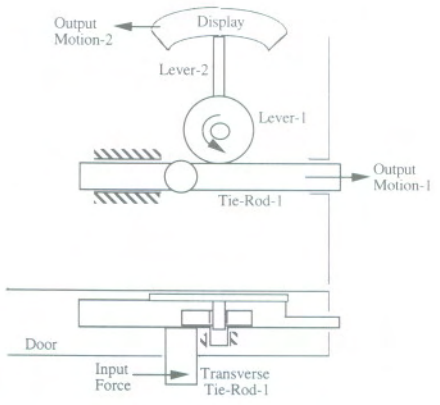


a. Another Kind Synthesized Solution to the Toilet Door Lock Problem

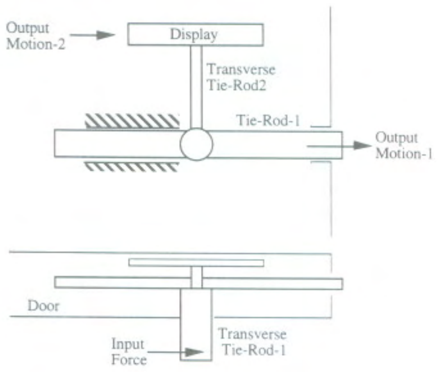


b. One Direction Configuration to the above Solution

Fig. 6. Another solution to the toilet-door lock problem.



a. A Schematic of the Solution Having Direction Configuration 2 in Fig. 4



b. A Schematic of the Solution Shown in Fig. 6

Fig. 7. Schematic representations of two different solutions generated by the synthesis procedures, to the toilet-door lock problem.

run on LispWorks™ (of Harlequin Ltd, UK). The procedures generate, not only the existing spatial configurations of a design, but also their other possible configurations. For instance, the solution in Fig. 2 can have not only the existing spatial configuration (see configuration 2 Fig. 4, and for a schematic of the solution, see Fig. 7a), but also, among others, an alternative one shown as configuration 1 in Fig. 4. This, together with the kind synthesis procedure, gives designers a two-fold flexibility in generating solutions. For instance, a different solution to the same problem and its one spatial configuration are shown in Fig. 6 (its schematic is shown in Fig. 7b).

The procedures are exhaustive within their scope. Appendix B provides a formulation of the complete set of configurations produced during the whole synthesis process (which includes both kind synthesis and the configuration procedures) for SISO systems. In Appendix C, a theoretical analysis of the performance of the overall synthesis procedure for SISO systems is provided. Appendix D provides a preliminary analysis of the performance of a configuration procedure, which shows that it is linear with respect to the size of the solution (i.e. the number of elements in the network representing the solution).

7. Conclusions and Future Work

Functional requirements of a mechanical transmission problem can be described in terms of the time-varying characteristics of inputs and outputs. These problems are proposed to be solved in three steps:

- Solve an instantaneous part of the functional requirements by primary structures
- Produce supporting structures required by these primary structures
- Carry out temporal reasoning to fit the solutions to the rest of the temporal requirements.

These three papers elaborate an approach to solve instantaneous requirements. The abstract descriptions of solutions so developed, if elaborated, will have the potential of producing designs, such as those shown in Fig. 7.

Future work involves the remaining two steps in the above problem-solving approach.

Acknowledgements

Amaresh Chakrabarti wishes to acknowledge the Nehru Trust for Cambridge University, India, Cambridge University Engineering Department, The trustees of the Lundgren Fund of Cambridge University, The Cambridge Engineering Design Centre, The Northbrook Society, Darwin College, The Gilchrist Educational Trust, and The Leche Trust, for financial support.

References

Chakrabarti, A., (1991) *Designing by functions*, PhD. Thesis, Department of Engineering, University of Cambridge.

Chakrabarti, A. and Bligh, T. P. (1992) A knowledge-based system for synthesis of single input single output systems in mechanical conceptual design. *Proc. 4th Int. Joint Euro-US Conf. on AI and Expert Systems Applications*, Houston, pages 401–406.

Clayton, B. D. (1985) *Inference ART Programming Manuals*, Inference Corporation, Califormia.

Appendix A: Pseudo-Code for the Orientation Configuration Procedure

The steps are:

Known:	A set of orientation-transformation operators for each of the structures constituting the solution concept considered (e.g., for shaft, (*iii*) (*jjj*) . . .)
Given Problem:	Transform, for the given solution concept, its: Orientation of one input → orientations of other Inputs and outputs

```
Step 1: Start the problem with
    (1) a list complete-list of structures (arcs) having specific input–output nodes (constituting the solution synthesized using kind synthesis);
    (2) a list of unpropagated-structures (a part of complete-list) through which orientation constraints have not yet been propagated;
    (3) a list of propagated-structures (another part of complete-list) through whose nodes orientation constraints have already been propagated; and ·
    (4) a list of nodes which should have a specified orientation value or values ($i/j/k$).
Step 2: When the list of unpropagated-structures is empty, terminate by returning the complete-list and the list of nodes as the result. Otherwise, do the following until the list of unpropagated-structures is empty:
    Step 2.1: Remove the first structure from the list.
        Option 2.1.1: If only one x-put (i.e., input or output) of the structure has an orientation value (o-value) in the nodes, compute the o-values of its other x-put from its orientation transformation, and keep it in the list of nodes. Keep the structure in the list of propagated-structures, and go back to Step 2.1.
        Option 2.1.2: If none of the x-puts of the structure have any o-value in the list of nodes, stack the structure at the back of the list of unpropagated-structures, and go back to Step 2.1.
        Option 2.1.3: If both the x-puts have o-values in the list of nodes, then:
            • If one of the x-puts is an intersection-value, calculate, using this value, the o-value of the other x-put. Otherwise, calculate the o-value of its output-node from the o-value of its input-node.
            • Find the intersection of the previous and the newly computed o-value of the x-put. Call it the intersection-value.
            • If there is no intersection, terminate the whole process by declaring no solution. Otherwise, if the intersection-value is the same as the previous and the computed values, keep the structure in the list of propagated-structures, and go back to Step 2.1. If the intersection-value is different from either of the above two values, do the following:
                • Change the o-value of this node in nodes to the intersection-value, and mark this as the present intersection-value.
                • Keep the structure in the list of propagated-structures.
                • Form a list of structures, from propagated-structures, which are connected to the node which has the present intersection-value.
                • Pass this list on to Step 2.1, and recur through Steps 2.1 onwards.
```

Appendix B: A Derivation of the Overall *Exhaustive* Set of Solutions Produced after the Kind Synthesis, Orientation Configuration and Sense Configuration of *SISO* Systems

Let the numbers of kind, orientation and sense variables be, respectively, k, o and s. The number of possible SISO transformers, therefore, are $(k.o.s)^2$. Each such transformer type is denoted by T_{ILO}, where I, L and O stand for input, length and output vectors respectively. Here, I and O can take any triplet value h-i-j, and L any doublet value i-j (where $h = 1, \ldots, k$, $i = 1, \ldots, o$, and $j = 1, \ldots, s$), where h, i, and j respectively denote kind, orientation and sense information. So in effect, L could take any value from the set $[1, \ldots, m; m = o.s]$, and I or O could take any value from set $[1, \ldots, n; n = k.o.s]$. N_{ILO} is used as the number of available different transformers of type T_{ILO}.

Suppose a SISO design problem is expressed as the following transformation:

$$a \rightarrow b$$

where a is the input variable, and b is the output variable. Let this problem be solved by using a maximum of r transformers. This is equivalent to forming chains of transformers, whose length should be, at the most, r.

For an exhaustive search, the number of solutions possible using a single transformer is all those which can take a as input and b as output. This is given by:

$$N(1) = \sum_{l(1)=1}^{m} N_{a.l(1).b} \tag{1}$$

The number of solutions possible using two transformers per solution is:

$$N(2) = \sum_{o(1)=1}^{n} \left[\sum_{l(1)=1}^{m} N_{a.l(1).o(1)} \sum_{l(2)=1}^{m} N_{o(1).l(2).b} \right] \tag{2}$$

The number of solutions possible using three transformers per solution is:

$$N(3) = \sum_{o(1)=1}^{n} \left[\sum_{l(1)=1}^{m} N_{a.l(1).o(1)} \times \sum_{o(2)=1}^{n} \left[\sum_{l(2)=1}^{m} N_{o(1).l(2).o(2)} \sum_{l(3)=1}^{m} N_{o(2).l(3).b} \right] \right] \tag{3}$$

By induction, the number of solutions using r transformers per solution is:

$$N(r) = \sum_{o(1)=1}^{n} \left[\left[\sum_{l(1)=1}^{m} N_{a.l(1).o(1)} \right] X \right], \tag{4}$$

where

$$X = \sum_{o(2)=1}^{n} \left[\sum_{l(2)=1}^{m} N_{o(1).l(2).o(2)} \cdots \sum_{o(r-1)=1}^{n} \left[\sum_{l(r-1)=1}^{m} N_{o(r-2).l(r-1).o(r-1)} \times \sum_{l(r)=1}^{m} N_{o(r-1).l(r).b} \right] \cdots \right] \tag{5}$$

So, the total number of solutions possible using a maximum of r^* transformers ($r \leq r^*$):

$$N(\textstyle\sum r^*) = \displaystyle\sum_{r=1}^{r^*} N(r), \text{ where } N(r) \text{ is given by Eq. (4).} \tag{6}$$

Therefore, for a given knowledge base (i.e., the n I/O variables, and the numbers N_{ILO} of available transformers of various types), and a given kind synthesis problem (i.e., the input variable a, output variable p, and the value of r^*), the size of the exhaustive set of SISO solutions can be obtained. Note that, if the various types of transformers available, for each specific input-output transformation required by Eq. (6), are put together in the sequences in which they are required by the equation, the SISO solutions themselves in the exhaustive set can be obtained.

Appendix C: An Analysis of the Effects of Available Knowledge Base, Allowable Number of Structures, and the Problem, on the Number of Solutions Produced in the Kind Synthesis of *SISO* Systems

The expression for the number of solutions possible, using r SISO transformers, for a given SISO kind synthesis problem, is given by (Eqs (4) and (5), Appendix A):

$$N(r) = \sum_{o(1)=1}^{n} \left[\left[\sum_{l(1)=1}^{m} N_{a.l(1).o(1)} \right] X \right], \tag{7}$$

where

$$X = \sum_{o(2)=1}^{n} \left[\sum_{l(2)=1}^{m} N_{o(1).l(2).o(2)} \cdots \sum_{o(r-1)=1}^{n} \left[\sum_{l(r-1)=1}^{m} N_{o(r-2).l(r-1).o(r-1)} \times \sum_{l(r)=1}^{m} N_{o(r-1).l(r).b} \right] \cdots \right] \tag{8}$$

To be able to visualize the behaviour of this equation, we need to remove the sensitivity of the equation with respect to specific problem and knowledge base situations. To do this:

Let $N \geq N_{ILO} \geq 0$, where N is the maximum size of N_{ILO}. So,

$$N_{ILO} = a_{ILO} . N \qquad \text{where} \quad 1 \geq a_{ILO} \geq 0.$$

Then,

$$N(r) = \sum_{o(1)=1}^{n} \left[\left[\sum_{l(1)=1}^{m} a_{a.l(1).o(1)} \right] X'' \right] N^r, \tag{9}$$

where

$$X'' = \sum_{o(2)=1}^{n} \left[\sum_{l(2)=1}^{m} a_{o(1).l(2).o(2)} \cdots \sum_{o(r-1)=1}^{n} \left[\sum_{l(r-1)=1}^{m} a_{o(r-2).l(r-1).o(r-1)} \times \sum_{l(r)=1}^{m} a_{o(r-1).l(r).b} \right] \cdots \right] \tag{10}$$

The coefficient to N^r in Eq. (9) is normalized by expressing the equation in the following form:

$$N(r) = \mathbf{a} n^{r-1} m^r N^r \tag{11}$$

where,

$$\mathbf{a} = \frac{1}{n^{r-1} m^r} \sum_{o(1)=1}^{n} \left[\sum_{l(1)=1}^{m} a_{a.l(1).o(1)} \cdots \sum_{o(r-1)=1}^{n} \left[\sum_{l(r-1)=1}^{m} a_{o(r-2).l(r-1).o(r-1)} \times \sum_{l(r)=1}^{m} a_{o(r-1).l(r).b} \right] \cdots \right] \tag{12}$$

and,

$$1 \geq \mathbf{a} \geq 0.$$

For a uniform knowledge base, i.e., where $N_{ILO} = N$, $\mathbf{a} = 1$. So, Eq. (11) becomes:

$$N(r) = n^{r-1} m^r N^r \tag{13}$$

For a given knowledge base (i.e., for fixed values of the n and N_{ILO}) and a given design problem (i.e., the values of r, input a, and output b are given, if $\mathbf{a} = 0$, the problem cannot be solved using that knowledge base. The conditions may be derived from the definition of **a** given above. Two evident ones are when $N_{aLO} = 0$ (i.e., there is no transformer which can take a as an input), or when $N_{ILb} = 0$ (i.e., there is no transformer which can deliver b as an output). The performance traits mentioned in Papers II and III are explained below.

1. The number of solutions, for a given problem using a given r, increases with the increase in the number of transformers in the knowledge base.

There are two cases when r is constant:

a. When a knowledge base is increased in terms of an increase of the values of the existing N_{ILO}s (i.e., m and n are not increased). In this case, only the value of **a** increases, thereby increasing the value of $N(r)$ in eq. (11).

b. When a knowledge base is increased by adding in new N_{ILO}'s values, without changing the existing N_{ILO}s (i.e., n and/or m is increased). In this case, although the value of $n^{r-1}m^r$ increases, the value of **a** may or may not increase. However, the value of the product $\mathbf{a}n^{r-1}m^r$ increases, thereby increasing the value of $N(r)$ in Eq. (11).

2. The number of solutions for a given problem and knowledge base increases exponentially with the increase in allowable r.

When the knowledge base is the same (i.e., m, n and N_{ILO}s are constant), $N(r)$ increases with r, as is evident from Eq. (11). This increase is exponential, i.e., proportional to $n^{r-1}m^rN^r$.

Appendix D: An Analysis of the Performance of the Constraint Propagation Procedure

As has been discussed in Section 2, a spatial configuration procedure (i.e., orientation or sense configuration procedure) contains essentially three types of operations. One is calculation of possible values (i.e., possible spatial arrangements) of the output vector and the length vector of an element, from given values of its input vector, for the first time (forward propagation). The second is modification of the existing value of an I/O node, if its newly calculated value is partially different from its existing value (node modification). The third type of operation is the back propagation of the effects of a node modification into those branches of the network (connected to the modified node) through which forward propagation has been done before (back propagation).

In order to calculate the maximum and minimum number of the above operations required for a given network with a given set of nodes with provided values (constraints) we proceed as follows:

The total number of clashes (a clash is the phenomenon of the procedure coming across the first node, during a forward propagation spell, which would require a modification) N_c is given by:

$$N_c = N_f - 1 \tag{14}$$

where N_f is the number of nodes with provided value.

Thus the total number of node modifications (same as the number of clashes) N_{nm} is given by:

$$N_{nm} = N_c = N_f - 1 \tag{15}$$

The total number N_{for} of forward propagation operations is the same as the number of elements in the network, and is given by:

$$N_{for} = N_e \tag{16}$$

The maximum number of back propagation operations required $N_{b-\max}$ after the first clash is the same as the length of the largest chain, and is given by:

$$N_{b-\max}\,(\text{after clash } 1) = N_e - N_f + 2 \tag{17}$$

where N_e: total no. of arcs in the network, which is one less than the total no. of nodes. The maximum number of back propagation operations required $N_{b-\max}$ after the rth clash is given by:

$$\begin{aligned} N_{b-\max}\,(\text{after clash } r) &= N_e - (N_c - r) \\ &= N_e - N_f + r + 1 \text{ (from (14)} \end{aligned} \tag{18}$$

The maximum total number of back propagation operations $\sum N_{b-\max}$ required:

$$\sum N_{b-\max} = \sum_{r=1}^{N_f-1} [N_e - N_f + r + 1]$$

$$\sum N_{b-\max} = \tfrac{1}{2}[N_f - 1][2N_e - N_f + 2] \tag{19}$$

The minimum number of back propagation operations required can be calculated by summing up the minimum number of such operations after each clash. As after the last clash, the effect of the node modification has to be back-propagated through the whole network, the number of operations after the last clash is given by:

$$N_{b-\min}\,(\text{after the last clash}) = N_e \tag{20}$$

As after each clash, there must be at least one element through which forward propagation has been done before, the minimum number of back propagations required after each clash except the last one is one. Thus the total minimum number of back propagations required is:

$$\sum N_{b-\min} = N_e + N_c - 1 = N_e + N_f - 2 \tag{21}$$

Equations (15), (16) (19) and (21) together describe the upper and lower bounds on the total number of (the three types of) operations required in a configuration procedure. One check to the correctness of these formulae is to check whether the maximum and minimum values of the number of propagations converge for the case where the number of operations is fixed. This is the case for a single chain, where the values for the two end-nodes of the chain is provided. In this case, for a chain having e elements (arcs), the above equations can be used to show that they converge:

$$N_e = e;\ N_f = 2;\ \Rightarrow N_{nm} = 1;\ N_{for} = e;$$
$$\sum N_{b-\max} = e;\ \sum N_{b-\min} = e.$$

As a general example, calculations for the case of the network in Fig. 1, using the above formulae, are given below:

$$N_e = 5;\ N_f = 4;\ \Rightarrow N_{nm} = 3;\ N_{for} = 5;$$
$$\sum N_{b-\max} = 12;\ \sum N_{b-\min} = 7;$$

this is justified by the actual number of operations for the case shown in Fig. 7, which are: $N_{nm} = 3$; $N_{for} = 5$; $\sum N_b = 11$. This is found by adding the number of operations for each type for all the stages in the process as shown in Fig. 1. For instance, the total number of back propagations performed is 11 of which two come from operations in stage 3, three from stage 5, and the rest from those in stage 7, in Fig. 1.

The equations (19) and (21) for a given problem (this means that the characteristics of required inputs and outputs are specified, and thus N_f is a constant), show that the number of operations increases linearly with the number of elements in the network.