Research in Engineering Design (1996) 8: 52–62

Research in
Engineering
Design

An Approach to Functional Synthesis of Solutions in Mechanical Conceptual Design. Part II: Kind Synthesis

Amaresh Chakrabarti and Thomas P. Bligh
The Engineering Design Centre, Department of Engineering, University of Cambridge, Cambridge, UK.

Abstract. *The three papers in this series 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 II, describes a set of procedures, which, using a knowledge base of primary structures expressed in terms of the representation constructs developed in Part I, can exhaustively synthesize topological descriptions of possible solutions to a given problem.

Keywords. Functional modelling; Functional reasoning; Functional synthesis; Mechanical design; Computer aided design; Transmission design; Concept generation; Topological synthesis; Exhaustive search

1. Introduction

Constructs for describing problems and solutions in transmission design were described in Section 7 of Part I (Chakrabarti and Bligh 1994), Introduction and Knowledge Representation. To recapitulate, a multiple input–output problem is described in terms of the characteristics of each of the required inputs and outputs. These characteristics include 'kinds' (forces and motions), 'orientations' (i, j or k), 'senses' (+ or −), 'magnitudes' (some real number), and 'positions' ($xi + yj + zk$, where x, y and z are real numbers). A 'primary structure' is described in terms of three vectors: an 'input vector', an 'output vector', and a 'length vector' connecting these two. An input or an output vector has all the characteristics mentioned above, while a length vector has all but the kind-characteristic. Each primary structure has a set of relationships between each characteristic of its vectors; these are called 'transformations'. A structure, therefore, has some 'kind-transformations', which describes the kinds of inputs and corresponding outputs permitted by the structure. A primitive lever, for instance, can take a force or linear velocity as input, and a torque or angular velocity as its corresponding output, and vice versa. For each of these kind transformations, there can be a number of 'orientation-transformations', each of which is a relation that gives a possible combination of orientation characteristics of the three vectors describing the structure. Similarly, for each such orientation-transformation, there can be a set of 'sense-transformations', and so on.

Given a knowledge base of primary structures whose above transformations are known, a design problem, described in terms of the representation constructs mentioned above, can be solved by synthesizing these structures. The problem considered here is to generate solutions to a transmission design problem at an instant of time. The approach is to solve the problem at each instant of time, following the order in which the I/O characteristics are enlisted in Section 7 of Part II (i.e., Kind, Orientation, Sense, Magnitude and Position requirements). We first consider the problem of 'kind synthesis' (i.e., synthesizing only those solutions which will provide the I/O kinds required by the design problem). We then evaluate each of the above solutions for orientation, thereby configuring the valid orientations of these solutions. These orientations are then checked for the sense requirements, and the valid sense configurations are computed and retained. These solutions are now ready to be evaluated for the magnitude and position requirements, again allowing the elimination of the infeasible solutions. This paper describes the kind synthesis procedures.

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

2. Kind Synthesis of Single Input–Single Output (*SISO*) Systems

The problem here is to devise a procedure that would synthesize solution concepts, using a knowledge base of structures and their rules of combination, to a given design problem of transforming an input of a given kind to an output of a desired kind. If for instance, we want to transform an input force into an output torque, the kind synthesis procedure should be able to generate concepts that would take a force as an input and provide a torque as an output.

This problem can be viewed as a 'search' problem, 'satisficing' (Simon, 1969) or exhaustive, where there are defined 'initial' (input kind) and 'goal' (output kind) 'states', and the problem is to move from the initial state to the goal state using valid 'operators' (known primary structures) that change the state of the problem. Each resulting solution should be a 'causal chain' of operators (i.e., primary structures) connecting the given input kind to the desired output kind. In this case, we try to solve the exhaustive search problem, of which searching for satisficing solutions is a sub-problem.

There are two questions. One is, how do we choose the first structure, and the next, and so on? The second is, where do we stop this process?

The rule of combination (which in this case is that two structures can be connected, if and only if the output kind of one structure is the same as the input kind of the other) provides the way of choosing succeeding structures at any state of the problem. We can choose, as the first structure, any structure from the knowledge base, whose input kind is the same as that of the specified problem input. The resulting state of the problem can now be given in terms of the output kind of the chosen structure, and the next operator can be chosen from the structures that can take, as input, the output kind of the previously chosen structure.

One answer to the second question is that the procedure would stop as soon as the problem state matches the goal state, i.e., the output of the most recently chosen structure matches with the specified problem output. However, as we are interested in an exhaustive set of solutions, the method is to explore every possible branch of the solution tree, until all the solutions are found. It may be that for a specific branch, it is not possible to find a solution by using a finite number of structures. For example, in a problem of transforming a force into a torque, we could to on transforming the input by using a series of force-to-force transformers, thereby never arriving at the torque. The other possibility is that a solution space can be potentially infinite, and therefore, can never be explored exhaustively in an absolute sense. For instance, a force-to-force transformation could be achieved by connecting one force-to-torque structure with its inverse, or by connecting four such structures, or eight, and so on. To eliminate these possibilities, the problem is constrained by specifying the maximum number of structures that can be used in spawning any branch.

Therefore, given an allowable set of structures having known kind-transformations (e.g., torque-to-torque transformation for a shaft), the kind synthesis procedure should be able to generate all the solutions which could transform the given input kind to the required output kind (e.g., torque-to-torque) using a maximum of r structures from the given set (e.g., 4 for the bicycle drive in Fig. 7 in Part I). The procedure is as follows (see Fig. 1):

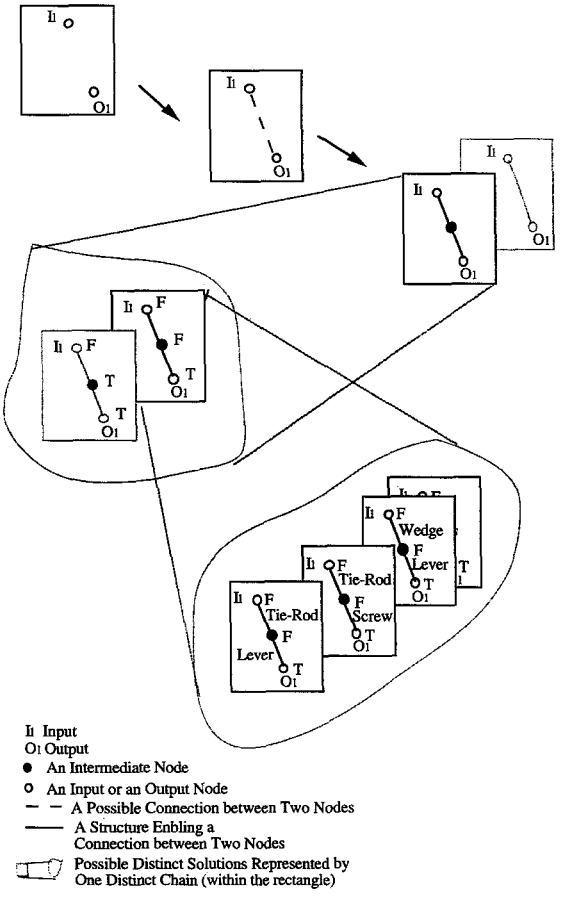


Fig. 1. Generating single-input single-output chains of structures (arcs) as conceptual solutions to a single-input single-output problem using a maximum of two structures; only one branch of the solution space is expanded here. The possible distinct solutions represented by one distinct chain is shown by expanding its content.

Known: A set of kind-transformation structures (i.e., structures such as shaft: torque ↔ torque; lever: force ↔ torque; etc.)

Given Problem: Find all the solutions which transform the given Input kind to the required Output kind (e.g., force → Torque)

Maximum number of structures allowed: r[1]

Step 1: Set the problem input as the *present input*. Set the problem output as the *required output*.

Step 2: List all the structures whose input matches the *present input*. For each such structure do the following.

Step 2.1: check whether or not the output of the structure matches the *required output*.

Option 2.1.1: It it matches, keep a copy of the chain of structures, used in the present branch, as a solution in a *list of solutions*. Check to see if the total number of structures used in the present branch is less than r.

Option 2.1.1a: If yes, set the output of the last structure used as the *present input*, and follow through Step 2 onwards.
Option 2.1.1b: If no, terminate the process by returning the *list of solutions*.

Option 2.1.2: If matching does not succeed, set the output of the last operator used as the *present input* and continue through Step 2 onwards.

3. An Example of Kind Synthesis of *SISO* Systems

Suppose we want to devise solution concepts to the problem of using a small hand force for unlatching a door, and we have already decided that the door would be latched by inserting a slider into a slot. Using the problem representation constructs that the synthesis procedure can recognize, we can represent this problem as a transformation between an input force (and associated motion) and an output linear motion. The function of the problem at an instant can be described as an instantaneous transformation between an input force and an output linear motion. This can

[1] The idea is that the designer would start synthesis with a small value of r, and would increase its value if this does not allow satisfactory solutions to be found.

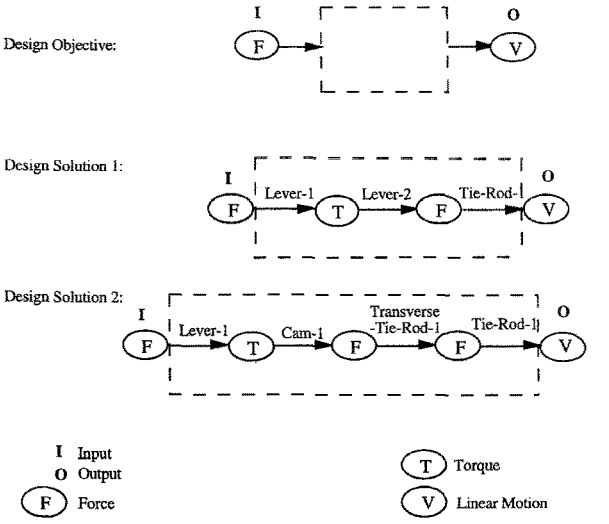


Fig. 2. A SISO kind synthesis example.

be specified by:

Input
Kind: force
Orientation: k
Sense: +
Magnitude: magnitude – 1
Position: $(x_1 i + y_1 j + z_1 k)$

Output
Kind: linear motion
Orientation: i
Sense: +
Magnitude: magnitude – 2
Position: $(x_2 i + y_2 j + z_2 k)$

Now, starting with the kind synthesis part of the problem, the output of the procedure would be a list of causal chains, each of which could be represented as a directed graph of a set of structures. For instance, given a suitable knowledge base, if r is set as 4 for this problem, two of the 255 solutions produced would be (see Fig. 2):

Solution 1: (lever-1 force → torque) (lever-2 torque → force) (tie-rod-1 force → linear motion)

Solution 2: (lever-1 force → torque) (cam torque → force) (transverse-tie-rod-1 force → force) (tie-rod-1 force → linear motion)

This synthesis procedure would require modification, if this were to apply to synthesizing single input multiple output (SIMO) or multiple input single output (MISO) systems, which is discussed next.

4. Kind Synthesis of *SIMO* and *MISO* Systems

Single Input–Multiple Output (SIMO) systems are in essence equivalent to Multiple Input–Single Output (MISO) systems. Discussion here is therefore limited to considering the synthesis of single input–multiple output systems. These problems, as in the synthesis of SISO systems, can be viewed as *exhaustive search* problems, where there are defined *initial* (input kind) and *goal* (output kinds) *states*, and the problem is to move from the initial state to the goal state using valid *operators* (known structures) that change the state of the problem. The resulting solutions should be *causal trees* of operators (i.e., primary structures) connecting the given input kind to the desired output kinds.

Therefore, given an allowable set of structures having known kind-transformations (e.g., torque-to-torque transformation for a shaft), the kind synthesis procedure should be able to generate all the solutions which could transform the given input kind to the required output kinds (e.g., a force to two torques) using a maximum of r structures from the given set. The working of the SIMO synthesis procedure can be viewed as an extention of the SISO synthesis procedure, and is illustrated for a single input–two output problem using $r = 4$ (see Fig. 3). Here each output is connected, using the SISO procedure, to one of the potential inputs (which can be either the input specified by the problem, or some intermediate input generated while forming the previous causal chains).

The kind synthesis procedure is as follows:

Known: A set of kind-transformation structures (e.g., shaft torque ↔ torque, lever force ↔ torque; etc.)

Given Problem: Find all the solutions which transform the given Input kind into the required Output kinds (i.e., input: force → output-1: torque, output-2: force).

Maximum number of structures allowed: r

Step 1: Start with a list of

(1) nodes that can be used as valid *input nodes*

(2) system *output nodes* that are not connected to any other nodes

(3) *connections* already produced (which connect two or more nodes)

(4) total *number* of structures (i.e., operators) that can be used to synthesize the solutions.

Step 2: Remove the first node from the list of unconnected *output nodes.*

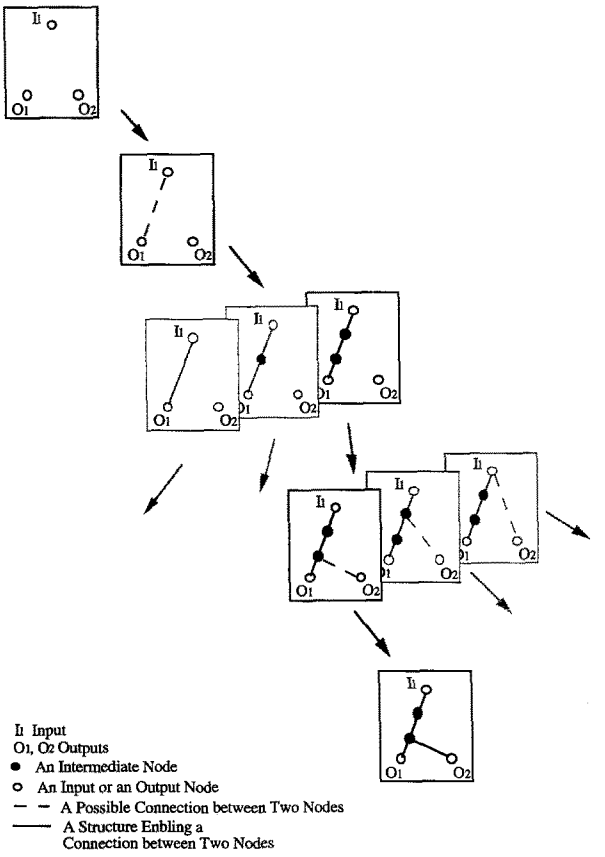


Fig. 3. Generating single-input multiple output networks of structures (arcs) as conceptual solutions to a single-input two-output problem using a maximum of four structures; only one branch of the solution space (highlighted) is expanded here.

Step 3: Prepare a list of arcs connecting each of the valid inputs in *input nodes* to this recently removed output node. Calculate how many structures can be used to form this connection (by subtracting from r the number of arcs in the list of *connections*, and the number of nodes in the present *output nodes*). If this number is zero, terminate. Otherwise, for each arc of this list, do:

Step 3.1: Use the SISO kind synthesis procedure to form a list of chains connecting the nodes that form the arc. For each such chain do:

Step 3.1.1: Modify the list of valid *input nodes* by including the newly generated intermediate nodes in the presently considered chain.

Step 3.1.2: Modify the list of already produced connections by including in the previous list the newly produced arcs.

Step 3.1.3: If the list of unconnected *output nodes* is empty, keep the already produced connections as one solution, and terminate. Otherwise, follow Steps 2 onwards.

5. An Example of Kind Synthesis of *SIMO* Systems

Suppose we want to devise solution concepts to the problem of using a small hand force for locking and unlocking a toilet door and providing a signal. There are two problems. One is *(un)locking* of the door, and the other, *indicating* to people outside that it is (un)locked. Suppose we have already decided that the door would be locked by inserting a slider into a slot, and the indication (whether it is locked or not) would be given by bringing into view some suitable signal (such as a colour code). Using the problem representation constructs that the synthesis procedure can recognize, we can represent this problem as a transformation between an input force (and associated motion) and two output linear motions (one for locking, and the other for indication). The instantaneous function of the problem at an instant can be described as an instantaneous transformation between an input force and two output linear motions, which can be specified by:

Input
Kind: force
Orientation: k
Sense: +
Magnitude: magnitude − 1
Position: $(x_1 i + y_1 j + z_1 k)$

Output-1
Kind: linear motion
Orientation: i
Sense: +
Magnitude: magnitude − 2
Position: $(x_2 i + y_2 j + z_2 k)$

Output-2
Kind: linear motion
Orientation: i
Sense: + or −
Magnitude: magnitude − 3
Position: $(x_3 i + y_3 j + z_3 k)$

Now starting with the kind synthesis part of the problem, the output of the procedure would be a list of causal trees, each of which could be presented as a directed graph of a set of structures. For instance, for a suitable knowledge base, if r is set as 4 for this problem, one of the 945 solutions produced would be (see Fig. 4):

One solution: (transverse-tie-rod-1 force → force) (tie-rod-1 force → linear motion) (lever-1 force → torque) (lever-2 torque → linear motion)

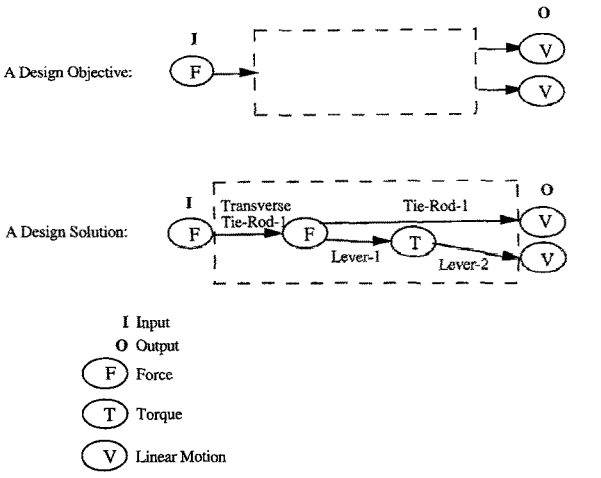


Fig. 4. A kind synthesis example: the toilet-door lock problem and one of its SIMO solutions.

The SIMO synthesis procedure needs modification in order to apply to synthesis problems involving multiple inputs and multiple outputs (MIMO); this is discussed next.

6. Kind Synthesis in *MIMO* Systems

As in the less general cases, Multiple Input-Multiple Output (MIMO) problems are also viewed here as (exhaustive) seach problems. The MIMO synthesis procedure is an extension of the SIMO synthesis procedure; here the outputs are first connected to some inputs, and then the remaining (unconnected) inputs are connected to some outputs, see Fig. 5 for an illustration. The procedure is:

Known: A set of kind-transformation structures (e.g., shaft: torque ↔ torque, lever: force ↔ torque; etc.)

Given Problem: Find all the solutions which transform the given Input kinds into the required Output kinds (e.g., input-1: force-1, input-2: force-2 → output-1: torque, output-2: force).

Maximum number of structures allowed: r

Step 1: Stack inputs, stack outputs.

Step 2: Connect each of the outputs to any of the inputs, directly or indirectly, in all possible ways, without repeating any solution.

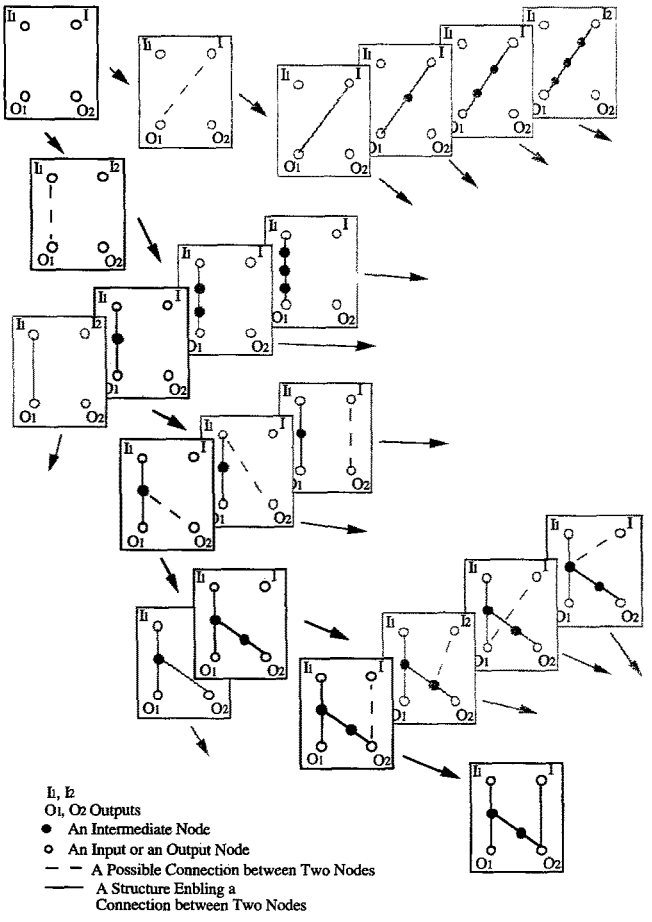


Fig. 5. Generating multiple input-output networks of structures (arcs) as conceptual solutions to a two-input two-output problem using a maximum of five structures: only one branch of the solution space (highlighted) is fully expanded here.

Step 3: Connect the remaining inputs (which were not connected in the above process) to any of the outputs, without generating any solution more than once.

Step 4: Among the solutions generated above, identify those, having more than one independent network, which have not used all the allowed operators. Connect these independent networks in all possible ways, using the maximum allowed number of spare operators.

Step 5: Among all the solutions generated in Step 4, identify those which still have not used all the allowable number of operators. Using these spare operators, connect nodes, within the same (independent) network in a solution, in all possible ways, without forming *circuits* (i.e., without joining two nodes using more than one path) and without repeating any solutions already generated.

7. An Example of Kind Synthesis of *MIMO* Systems

We would like now to re-examine the toilet-door lock problem, discussed in the SIMO synthesis example Section 5, with the following changes. Instead of a single input, the specification now calls for two equal, opposite and non-collinear input forces, i.e., a couple. Using the problem representation constructs that the MIMO synthesis procedure can recognize, we can represent this problem as a transformation between two equal and opposite input forces (and associated motion), and two output linear motions (one for locking, and the other for signalling). The function of the problem at an instant can be described as an instantaneous transformation between two input forces and two output linear motions. A complete specification of this problem would be:

Input-1
Kind: force
Orientation: k
Sense: +
Magnitude: magnitude – 1
Position: $(x_1 i + y_1 j + z_1 k)$

Input-2
Kind: force
Orientation: k
Sense: –
Magnitude: magnitude – 2
Position: $(x_2 i + y_2 j + z_2 k)$

Output-1
Kind: linear motion
Orientation: i
Sense: +
Magnitude: magnitude – 3
Position: $(x_3 i + y_3 j + z_3 k)$

Output-2
Kind: linear motion
Orientation: i
Sense: –
Magnitude: magnitude – 4
Position: $(x_4 i + y_4 j + z_4 k)$

Now, starting with the kind synthesis part of the problem, the output of the procedure would be a list of causal networks, each of which could be represented as a directed graph of a set of structures. For instance, for a suitable knowledge base, if r is set as 5 for this problem, one (see Fig. 6) of the ($\approx$10000) solutions produced would be:

One solution:
(lever-1 input force-1 → intermediate torque)
(lever-2 input force-2 → intermediate torque)

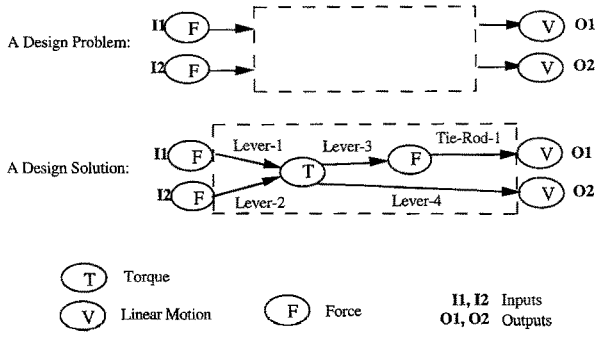


Fig. 6. A MIMO kind synthesis example: the toilet-door lock problem and one of its MIMO solutions.

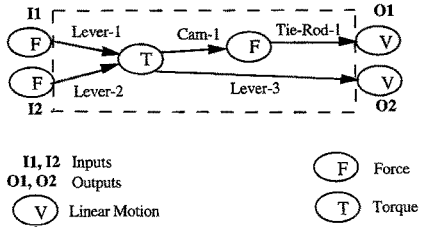


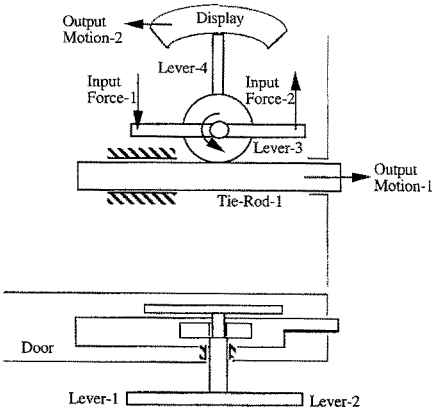
Fig. 7. Another solution to the toilet-door lock problem shown in Fig. 6.

(lever-3 intermediate torque → intermediate force)
(lever-4 intermediate torque → output linear motion-2)
(tie-rod-1 intermediate torque → output linear motion-1)

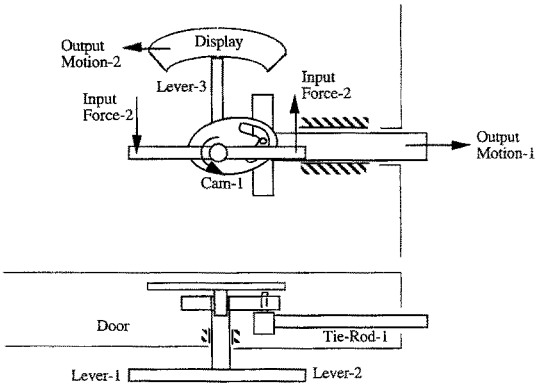
8. Implementation and Validation

The knowledge base described before, and the kind synthesis procedures, are implemented on a LispWorks™ (Harlequin Ltd, UK) environment. The language used is Common-LISP.

In MIMO systems synthesis, as well as in SISO and SIMO systems synthesis, it is found that, by using the knowledge of kind transformation of the primary structures *extracted* and *represented* from solutions to a set of problems (described in Section 7 of Part I) and by using the procedures described in Section 6 of this paper, it was possible to synthesize the graph-structures of these solutions (and more), to their corresponding problems. For instance, the procedure, implemented in the MIMO synthesis programs, generates, for the toilet-door lock problem (Fig. 6), not only the existing solution but also other feasible solutions such as the one shown in Fig. 7 (see Figs 8a and 8b for schematic diagrams of these solutions). This was also the case for the other problem–solution pairs which were used in the knowledge extraction phase.



(*a*) A schematic of the solution shown in Fig. 6.



(*b*) A schematic of the solution shown in Fig. 7.

Fig. 8. Schematic representations of an existing solution and a new solution generated by the MIMO synthesis procedures, to the toilet-door lock problem.

9. Evaluation of the Procedures

These procedures can be evaluated by asking two questions:

- Does the procedure serve its purpose?
- How effectively does it serve its purpose?

Within the scope of the problems discussed here, the synthesis procedures were expected: (1) to produce all the possible solutions (i.e., they are supposed to be *exhaustive*), and (2) to be able to do this is in a reasonable time. The number of feasible solutions possible, for a given problem, depends on the following parameters:

(1) The number and characteristics of the inputs and outputs of the problem;
(2) The number and characteristics of the transformers available in the knowledge base;

(3) The maximum number of structures that can be used in constructing the solution concepts.

Assuming that the available knowledge base would contain transformers of sufficient types, so as to transform an input of any allowable kind to an output of any allowable kind, i.e., when the knowledge base is 'complete', according to Prabhu and Taylor (1988):

- The number of solutions possible, for a given problem using a given r, always increases with an increase in the number of transformers available in the knowledge base.
- The number of solutions possible increases exponentially with an increase in r, for a given problem and knowledge base.

In order to evaluate the performance of the procedures, we need to check, (i) whether or not the procedures produce all the feasible solutions to a given problem, using no more than r transformers from a given knowledge base (i.e., the *exhaustiveness*), and (ii), whether this solution-set is produced in a reasonable time (i.e., the *time-effectiveness*).

9.1. Exhaustiveness

There are two ways of proving the exhaustiveness of a procedure A. One is to prove it mathematically. The other is to find another procedure B which yields the same result as that of procedure A under the same situations, and can be proved to be exhaustive. The second method is adopted here to substantiate the exhaustiveness of the synthesis procedures. Procedure B follows these steps:

(a) Enumerate all possible networks that can connect the given input(s) to the given output(s) of the problem.
(b) Label the nodes of each network in all possible ways, using variables (in mechanical design, these are force, torque, etc.) from the set of variables available in the knowledge base.
(c) For each possible network (or a set of networks) having a distinct set of labeled nodes, calculate the total number of possible solutions, by labeling the arcs (i.e., operators such as lever, shaft, etc.) in all possible ways, using available structures in the knowledge base as possible arcs.
(d) Add them to find the total number of solutions.

In the case of the synthesis of SISO systems, each of the steps are shown to be exhaustive, and hence the procedure is exhaustive (see Appendix A). In the case of SIMO/MISO systems synthesis, although steps (b) and (c) were exhaustive, and the concept of symmetry was used, manually, to check the exhaustiveness of the networks produced (i.e., step (a)) for the cases involving few inputs, outputs and maximum allowable operators, no formal proof is yet available.

In the case of MIMO systems synthesis, step a was manually carried out, using methods used in substantiating the exhaustiveness of the SISO and SIMO/MISO synthesis procedures, for small problems and was found to be exhaustive. However, a general procedure for carrying out step (a) remains to be found.

9.2. Time-Effectiveness

The procedures solve a problem in exponential time, i.e., the amount of time required to solve a problem is linearly proportional to the size of its solution-set which increases exponentially with r, for a given knowledge base and problem, and always increases with an increase in the number of transformers in the knowledge base, for a given problem and r. Some results illustrating the performance of the SISO synthesis algorithm are provided in Appendix B. An elaborate analysis of these effects, for SISO systems, is provided in Part III (on Spatial Configuration) of this paper series.

9.3. Combinatorial Explosion

As discussed, the procedures generate an exhaustive set of feasible solutions to a given problem. This solution-set is large, and increases exponentially with r and knowledge base. One way of solving the combinatorial problem is to force the solution-set to be small (possibility 1). If the efficiency of the algorithm can be increased, problems of larger size than at present could be tackled (possibility 2). Another method, which is not considered here, is to incorporate (comparative) evaluation of each solution as it is being produced, thereby identifying the optimal solutions rather than feasible solutions. This will require a second theory for (comparative) evaluation by which the quality of a complete solution could be judged by considering only a part of it (possibility 3). In the absence of this, either 'heuristically based' or 'near-optimal' algorithms have to be considered (possibilities 4 and 5). A long-term possibility might be the advancement of parallel processing in computers (possibility 6). These possibilities are elaborated below:

(1) Attack the parameters that contribute to the combinatorial problem:
Keep the required knowledge base and r

manageably small. When r is small, 'basic' (Chakrabarti, 1991) structures can be used. On the other hand, if a large r is needed when using basic structures, use compound structures (macros) of the relevant degree of detail which will reduce r. For example, use pumps, motors and valves as macros, rather than systems of basic structures, when designing a plumbing system.

(2) Devise more efficient algorithms:

Improve the algorithm having 'exponential complexity' into one that has 'sub-exponential complexity' (such as changing a 2^n algorithm into a $2^{n/2}$ algorithm). This will increase the maximum size of problems that can be solved in a reasonable time.

(3) Use sound theories for evaluation:

A sound theory for (comparative) evaluation, which can be used to judge how good the complete solution will be, by examining an incomplete solution, would allow for rapid elimination of incompetent designs at an early stage of their generation, thereby reducing the changes of combinatorial explosion.

(4) Use 'Heuristics':

Use 'heuristics' (judgement criteria, which can usually enable the algorithm to quickly prune the solution space, but is not guaranteed to work) in an existing algorithm in those cases where they work. Research is required in order to identify these heuristics.

(5) Look for 'near-optimal' algorithms:

- Instead of requiring an algorithm to always generate the optimal solution, it might be relaxed so that the algorithm must always generate a feasible solution with a value 'close' to that of an optimal solution (this is called an 'approximate' solution, and the algorithm which generates it is called an 'approximate' algorithm).
- Look for an algorithm which almost always generates an optimal solution. Algorithms with this property are called 'probabilistically good' algorithms.

(6) Change the technology:

With an increase in speed and parallel processing and decrease in the cost for arithmetic and logical operations, it should be possible to tackle problems of much larger size (possibilities 2, 4, 5 and 6 are discussed at greater length in Horowitz & Sahni, 1978).

10. Summary and Conclusions

Within the framework described in Part I (Introduction and Knowledge Representation), this part describes the step-by-step development of procedures for kind synthesis of up to multiple input–multiple output systems. The procedures use a knowledge base of primary structures, such as functional descriptions of levers and shafts, to produce networks of these structures as solutions to transmission problems described in terms of their inputs and outputs. The knowledge base, design problems, and solutions are expressed in terms of a set of representation constructs descibed in Part I. The solutions produced by the procedures include existing solutions as well as feasible new ones. The procedures generate an exhaustive set of solutions within the scope of the knowledge base used, and have combinatorial problems. However, these procedures are rudimentary in the sense that they do not use any heuristics or constraints apart from the kind-requirements of the problem inputs and outputs. It is hoped that judicial inclusion of heuristics and other methods mentioned in this article would improve their performance.

Acknowledgements

Amaresh Chakrabarti wishes to acknowledge the Nehru Trust for Cambridge University, India, Cambridge Philosophical Society, 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

Horowitz, E. and Sahni, S. (1978) *Fundamentals of Computer Algorithms.* Pitman Publishing Ltd, London.

Prabhu, D. R. and Taylor, D. L. (1988) 'Some issues in the generation of the topology of systems with constant power-flow input-output requirements'. *Proc. of The ASME Design Technology Conferences—The Design Automation Conference*, Rao, S. S., editor), Kissimmee, Florida, September, pages 41–48.

Simon, H. A. (1969) *The Sciences of the Artificial*, The MIT Press, Cambridge MA.

Chakrabarti, A. (1991) *Designing by functions*, Chapter 7, PhD Thesis, Department of Engineering, Cambridge University, UK.

Chakrabarti, A. and Bligh, T. P. "An approach to functional synthesis of solutions in mechanical conceptual design: Part I: introduction and knowledge representation." *Research in Engineering Design*, 1994, 6: 127–141.

Appendix A: A Derivation of the *Exhaustive* Set of Solutions Produced in the Kind Synthesis of *SISO* Systems

For the n possible *kind-variables* (such as force, torque, etc.) permitted to be considered for a design, there can be n^2 possible SISO transformers. Each such transformer type is denoted here by T_{ij}, where i denotes the input kind and j denotes the output kind of the transformer ($i = 1, \ldots, n; j = 1, \ldots, n$). Similarly, $N_{i \cdot j}$ is used as the number of available different transformers of type T_{ij}.

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

$$k \rightarrow p$$

where k is the input kind-variable, and p is the output kind-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 k as input and p as output. This is given by:

$$N(1) = N_{k \cdot p} \tag{1}$$

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

$$N(2) = \sum_{i(1)=1}^{n} [N_{k \cdot i(1)} N_{i(1) \cdot p}] \tag{2}$$

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

$$N(3) = \sum_{i(1)=1}^{n} \left[N_{k \cdot i(1)} \sum_{i(2)=1}^{n} [N_{i(1) \cdot i(2)} N_{i(2) \cdot p}] \right] \tag{4}$$

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

$$N(r) = \sum_{i(1)=1}^{n} \left[N_{k \cdot i(1)} \sum_{i(2)=1}^{n} \left[N_{i(1) \cdot i(2)} \cdots \sum_{i(r-1)=1}^{n} [N_{i(r-2) \cdot i(r-1)} N_{i(r-1) \cdot p}] \cdots \right] \right] \tag{4}$$

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

$$N(\Sigma r^*) = \sum_{r=1}^{r^*} \sum_{i(1)=1}^{n} \left[N_{k \cdot i(1)} \sum_{i(2)=1}^{n} \left[N_{i(1) \cdot i(2)} \cdots \sum_{i(r-1)=1}^{n} [N_{i(r-2) \cdot i(r-1)} N_{i(r-1) \cdot p}] \cdots \right] \right] \tag{5}$$

Therefore, for a given knowledge base (i.e., the n kind-variables, the numbers $N_{i \cdot j}$ of available transformers of various types), and a given kind synthesis problem (i.e., the input variable k, 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. (5), 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 B: Some Results of the Performance of the *SISO* Algorithm

Figures 9 and 10 are two scatter-plots of the number of solutions produced (N), for the same problem, using different values of the number of elements used in the solution (r), and different knowledge bases. Each set of data-points, having the same symbol, represents the typical effect, on the number of solutions produced (N), of the allowable number of elements (r), for a given knowledge base. Data-points having separate

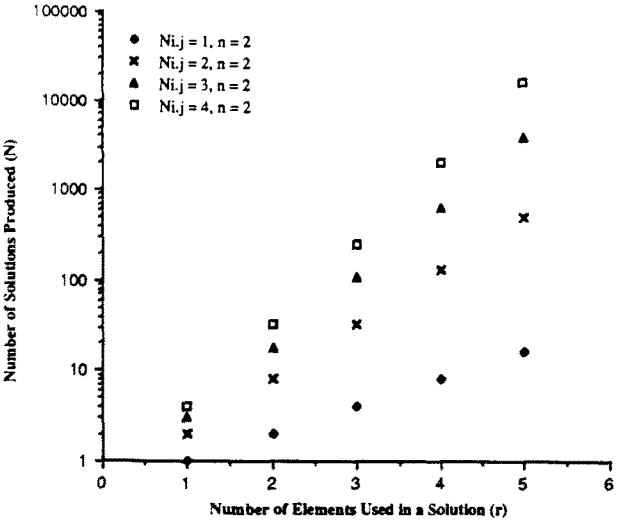


Fig. 9. Scatter-plot showing number of solutions N produced for various values of maximum allowable number r of elements for a knowledge base with a constant n and a variable $N_{i \cdot j}$.

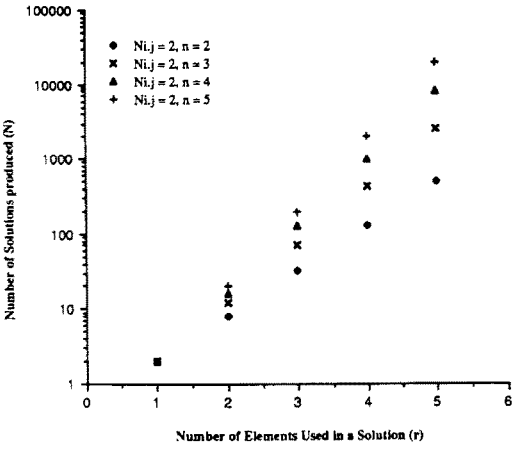


Fig. 10. Scatter-plot showing number of solutions N produced for various values of maximum allowable number r of elements for a knowledge base with a variable n and a constant $N_{i\cdot j}$.

symbols shows the above change for separate, and as one moves up, for increasingly larger knowledge bases. The size of a knowledge base depends on two parameters. One is n^2, the number of different transformer-types that are available in the knowledge base (such as force-to-force transformers, and torque-to-force transformers); the other is the number of different transformers (solution-elements) $N_{i\cdot j}$ available for each such type (such as a screw and a cam, both of which do a torque-to-force transformation). The total number of different solution-elements is given by

$$\sum_{i=1}^{n} \sum_{j=1}^{n} N_{i\cdot j}$$

The number of solutions, for a uniform knowledge base, where $N_{i\cdot j}$ for any i and j has the same value, would be $N_{i\cdot j} \times n^2$. Datapoint-sets in Fig. 9 are plotted for a constant n, and an increasing $N_{i\cdot j}$; Figure 10 is a plot for a constant $N_{i\cdot j}$ and an increased n for each data-set. Note that the number of solutions increases exponentially with r (linear change in a semi-log plot), and increases always with an increase in the knowledge base. For a non-uniform knowledge base, the number of solutions produced would lie between the bounding data-sets denoting the maximum and minimum values of its $N_{i\cdot j}$s. It is to be noted that all the above plots are solution-sets to a single problem, i.e., a single input–output problem. Similar plots can be obtained also for multiple input–output problems having various values in inputs and outputs. With the increase in the number of inputs and outputs, the number of solutions produced have similar characteristics, except that it increases faster with r and knowledge base.