Artificial Intelligence for Engineering Design, Analysis and Manufacturing (2000), *14*, 403–414. Printed in the USA.
Copyright © 2000 Cambridge University Press 0890-0604/00 $12.50

Increasing efficiency of compositional synthesis by improving the database of its building blocks

AMARESH CHAKRABARTI
Engineering Design Centre, Department of Engineering, University of Cambridge, Trumpington Street, Cambridge CB2 1PZ, U.K.

(Received October 5, 1998; Accepted April 5, 2000)

Abstract

This article is an attempt to improve the efficiency of procedures for compositional synthesis of design solutions using building blocks. These procedures have found use in a wide range of applications, and are one of the most substantial outcomes of research into automated synthesis of design solutions. Due to their combinatorial nature, these procedures are highly inefficient in solving problems, especially when the database of building blocks for synthesis or the problem size is large. Previous literature often focuses on improving only the algorithm part of a procedure, while it is both its algorithm and database which together determine the overall efficiency of the procedure. This article uses a case study to investigate and develop a set of rules for structuring and preprocessing a database of building blocks so as to improve the efficiency of synthesis of design solutions using this database.

Keywords: Automated Compositional Synthesis; Concept Generation; Database Preprocessing; Efficient Bidirectional Search; Engineering Design

1. INTRODUCTION

Compositional synthesis, where a set of building blocks is composed into networks as solutions for design problems, has been attempted for various domains of application (Pahl & Beitz, 1984; Prabhu & Taylor, 1988; Hoover & Rinderle, 1989; Ulrich & Seering, 1989; Finger & Rinderle, 1990; Hoeltzel & Chieng, 1990; Kota & Chiou, 1992; Malmqvist, 1993; Chakrabarti & Bligh, 1994, 1996*a*, 1996*b*; Welch & Dixon, 1994; Sushkov et al., 1996). Each of these systems requires a database of building blocks. Building blocks are simpler, constituting elements of the solutions found in the domain of application. For instance, Ulrich and Seering (1989) use generic physical systems elements, represented using bond graphs (Paynter, 1961), Hoover and Rinderle (1989) use gear-pairs, Chakrabarti and Bligh (1994) use motion elements, and Kota and Chiou (1992) use kinematic pairs, represented as matrices, as building blocks. The algorithms are essentially combinatorial in nature, often with a "generate and test" flavor.

In some earlier papers, an algorithm was proposed for exhaustive compositional synthesis of design solution principles in mechanical (Chakrabarti & Bligh, 1996*c*) and mechatronic (Chakrabarti et al., 1997) domains. A comparison between the solutions generated by the program and those considered independently by the designers (Burgess et al., 1995) in a case study showed that the computer suggested a wider range of interesting principles than designers considered on their own. This demonstrated the potential of the procedure in supporting designers' creative potential by exposing them to new solutions.

The two parameters of a synthesis procedure that researchers should be most concerned with are its *effectiveness* and *efficiency*. *Effectiveness* is defined here as the capability of generating new and interesting concepts, and has been dealt with elsewhere (Chakrabarti, 1998). One problem with compositional synthesis is its combinatorial nature, which makes it inefficient in enumerating a comprehensive range of solutions. The importance of having a large database of wide-ranging building blocks has long been recognized as useful for generating new and interesting concepts, and work is in progress in several groups towards developing such databases (Roth, 1970; Selutsky, 1987; Ishii et al., 1994; Tsourikov, 1995; Taura et al., 1996; Sushkov et al., 1996; Chakrabarti et al., 1997; Khang, 1998). However, efficiency of a synthesis procedure, which is defined here as the inverse of computational effort (i.e., time and memory) required for the procedure to generate a given solution set,

Reprint requests to: Amaresh Chakrabarti, Engineering Design Centre, Department of Engineering, University of Cambridge, Trumpington Street, Cambridge CB2 1PZ, U.K. E-mail: ac123@eng.cam.ac.uk

becomes particularly important if it must do so using these large databases.

A synthesis procedure has two parts: a database of building blocks and an algorithm that uses this database for synthesis. Previous efforts to improving efficiency have focused primarily on improving the algorithm. We wish to improve the efficiency of both the algorithm and the database to effect an overall improvement in the procedure. In a previous article (Chakrabarti, 1999), an attempt to improve the algorithm was reported. Substantial improvement in efficiency was achieved by using bidirectional instead of unidirectional search. However, the database used by such algorithms can still be poorly disposed towards solving a design problem, and thus there may be room for improving further the performance of a procedure by improving the quality of its database, an area hardly researched before.

Using a case study, this article investigates the role of the database of building blocks in the effectiveness and efficiency of a compositional computational synthesis procedure, and uses the results of this investigation to develop an approach for structuring and preprocessing a database so as to improve the efficiency of synthesis using this database.

2. DESIGN PROBLEMS CONSIDERED

The design problems considered here are those that can be expressed using a function transforming a given input into a required output. For instance, a sensing problem can be expressed in terms of the input signal to be sensed (e.g., acceleration in the case of an accelerometer problem), and an output medium (Chakrabarti et al., 1997) in which this signal is to be sensed (e.g., an electrical voltage). In the medical device domain, a drug infusion problem could be expressed in terms of an input signal leading to an output effect of a drug flow of some amount. In a mechanical device domain, an example would be a door locking problem whereby a given input motion of a door handle leads to a retraction motion of the latch. Although not all design problems can be expressed using inputs and outputs, as noted by Umeda and Tomiyama (1997), this representation is useful in a large variety of problems in many domains of application. The common form for these functions is shown in Figure 1.

3. DESIGN SOLUTIONS CONSIDERED

A design solution (also referred to as a solution principle) within the scope of this research is one that can be expressed as a composition of building blocks so that they are connected via their inputs and outputs to transform the given input of the design problem into its required output. Each building block therefore has an input and an output, and the building blocks constituting a solution principle transform the given input into a number of intermediate input/output variables before producing the required output. An accelerometer solution principle, for instance, can be a composition of three building blocks, an *inertia* block to transform the input acceleration into an inertia force, a *spring* building block to transform this force into a change in position, and a *capacitance* block to alter the voltage across a capacitor as a result of a change in capacitance due to the position change. A building block, in the context of the door locking problem, could be a *cam* block transforming the rotation of the handle into a translational motion and a *tie-rod* block transferring this motion to a different output position. A large number of design solutions in existence, as well as a substantial number of computer programs for design synthesis, are compositional in nature (e.g., Hoover & Rinderle, 1989; Ulrich & Seering, 1989; Kota & Chiou, 1992). Therefore, solutions of this kind (see Fig. 2) are fairly generic in their use in research and applications.

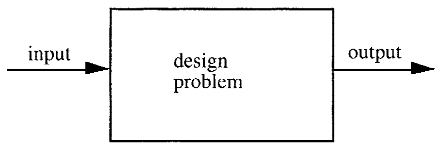


Fig. 1. The general form of design problems under consideration.

4. DATABASE OF BUILDING BLOCKS

A *database* is described here as a set of links between a set of nodes. Nodes are input/output parameters, for example, acceleration and voltage in the sample database shown in Figure 3. Directed links between the nodes (shown by arrows in Fig. 3) represent the building blocks (i.e., physical devices and effects capable of transforming the parameter from which the link starts, called *input*, into that in which it ends, termed its *output*). Each building block therefore has an input and an output node.

5. SYNTHESIS ALGORITHMS

The synthesis approaches have been described in detail in Chakrabarti (1996*a*). Here is a brief summary. The approach is to concatenate a set of building blocks from a database to form chains of building blocks, described here as solution principles, that transform the input parameter (signal to be sensed in the case of sensing problems) into the required output parameter (medium in which it is to be sensed). In this definition, an acceleration sensing problem would be described using acceleration as an input, and current or voltage as the output. A solution principle uses one or a series of building blocks to transform the input into the required output. If a database is described, as above, as a network of directed links between a set of nodes, and two of its nodes are described as the input and output required of a given design problem (in the case investigated these are acceleration and voltage, respectively), then each solution generated would be one possible distinct route between the input and output node. The way in which these routes

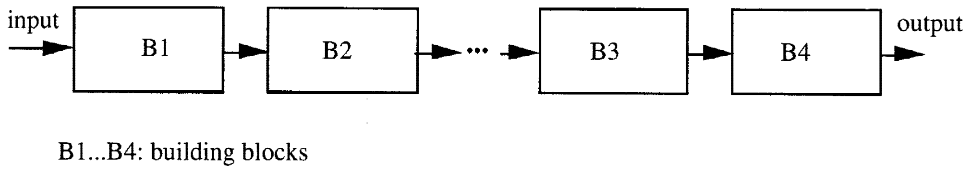


Fig. 2. The general form of the design solutions considered.

are identified constitutes the synthesis algorithms. Unidirectional search does this by progressively moving from the input towards the output node, whereas bidirectional search does this by progressing from both input and output nodes towards each other.

6. A CASE STUDY

In this case study, a database of building blocks was created by populating it with common building blocks identified by analyzing various existing sensing devices. We call this *random population* of the database, as the population process is not directed towards solving any specific synthesis problem. The synthesis approaches, mentioned in Section 5, were applied to this database to generate solution principles to an acceleration-sensing problem. These principles were then analyzed to see whether they were worth generating, and what caused their generation, so as to help encourage or avoid their generation. The solutions generated were evaluated for their worth by designers involved in the project.

6.1. Analysis of solutions

The analysis of the solutions identified five distinct features: 1) some solutions differed from others in terms of how they solved the major part of the required function; 2) some differed only in terms of how they solved the minor part of the function; 3) some were identical except for the granularity with which they were described, or 4) in terms of the detailed differences as to how their transformations were implemented, and 5) some of the building blocks in the database were not useful in solving the given problem.

6.1.1. *Major and minor parts of the function*

The major part of the function of an accelerometer is how to get some known electrical output as a measure of acceleration, and the minor part is how to transform this electrical output into a representative voltage. In this case study, although many principles generated by the computer solved the major part of the function using a wide variety of effects, many others were hardly different in the way they solved this major part, although they solved the minor part using a variety of ways.

A designer's objective would naturally be to maximize the variety of principles for the major function, and to avoid wasting time on finding creative alternatives for a minor part of the function. However, what is a major part for a particular design case may well be the minor part in another. Therefore, it is hard to specify what is major in a generic database. The strategy adopted is for the designer to decide the major part of the function on a case-by-case basis, and solve this part with the help of synthesis. The route envisaged is to allow designers the opportunity of dealing with more than one alternative output variable transforming to any of which will suffice as a solution to the problem.

6.1.2. *Principles which were different in terms of the effects they used*

In this category, the principles were different in terms of the underlying physical effects they used. For instance, contrast a piezo-type principle (see the top chain in Fig. 4) with a strain-gauge-type principle (bottom chain in Fig. 4). The piezo type uses *inertia* to transform the input acceleration

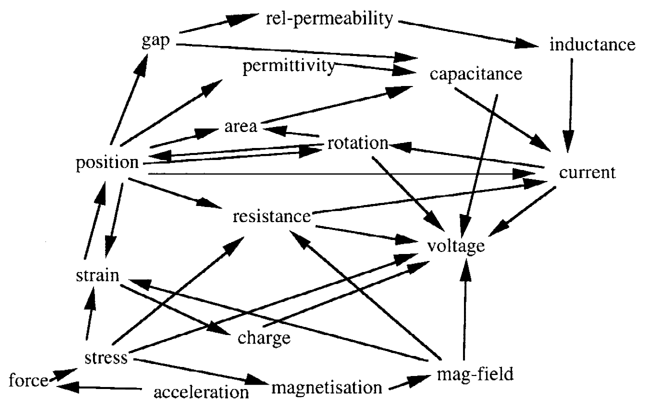


Fig. 3. A sample database having 18 nodes and 35 links (building blocks).

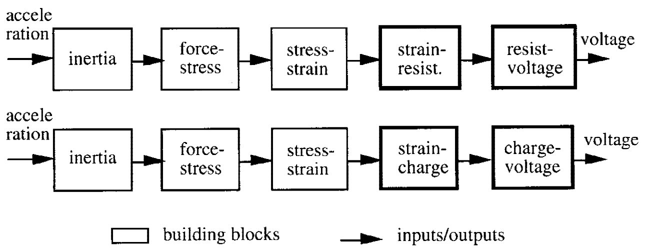


Fig. 4. The same transformation achieved by different building-block chains.

into a force, a *force–stress* building block to transform the force into a stress, and a *stress–strain* block to transform this stress into a strain, just as the strain-gauge type does. In the piezo case, this strain is then converted into an electrical charge using a piezo effect, followed by a change of the charge into a voltage. Contrast this with the strain-gauge type, where the strain, by definition, causes a length deformation leading to a change in resistance, which changes voltage using Ohm's law. The two principles, therefore, use a separate chain of physical effects to do the same transformation: of strain to voltage. We must retain the capability of generating principles of this kind. The wider the range of variables addressed by the building blocks in a database, and the more connected they are (via building blocks that can transform one variable to another), the more will be the variety of alternative principles generated by using such a database. We tried to attain this by ensuring a large number of variables shared by the building blocks in the database and by a large number of relationships between them (see Chakrabarti, 1998 for further details).

6.1.3. *Principles that are different only in terms of their granularity*

Consider the effect of generating a position change as a response to a force change. We could do this using a spring (top part, Fig. 5), which describes this as a *force-to-position* transformation, or we could do this using a combination of a *force-to-stress*, *stress-to-strain*, and a *strain-to-position* transformation (bottom part, Fig. 5), which is fundamentally how a spring works. If we keep all four of these building blocks in the database, we run into the danger of duplication. Keeping only the force–position block does not allow its constitutive building blocks to be used in other principles.

This is a good example of how efficiency and effectiveness are related. Avoiding duplication saves time, as the duplicates need not be generated, but how it is avoided determines whether such efficiency comes at the expense of effectiveness. It is decided that only building blocks that are more basic will be kept in the database so as to avoid compromise in effectiveness, for example, the building blocks force-to-stress, stress-to-strain, and a strain-to-position transformation rather than the monolithic spring in the above case.

6.1.4. *Principles that use alternative single building blocks to do the same transformation*

Take the example of transforming a rotation into a current. This can be done using two alternative building blocks: either by *Wiedemann* effect, or by an *eddy-current*-based configuration. Such distinctions are useful at a later stage, where a designer may wish to explore in more detail principles that realize this transformation. The database therefore should be maintained such that only one building block between any two variables is used, even if the block may have alternative incarnations in reality. This is done to avoid duplication of solution principles, at this stage, in terms of their possible detailed realizations.

6.1.5. *Building blocks that were not useful in solving a given problem*

By comparing the building blocks shared among the solutions generated with those in the database, it was found that several in the database were not used in any solutions. As a database is described here as a network of relationships among a set of variables, and a given problem is described in terms of two of its nodes (one as input and the other as output, e.g., acceleration and voltage, respectively), the solutions expected to be generated using the database are given by the routes that are possible between these two nodes of this network. Therefore, there can be nodes and links in the network that are not part of these routes, and therefore, not useful in solving this particular problem. For instance, generation of solutions to fulfill a function of transforming I (input) to O (output) will not require any of the links in the database shown in Figure 6 other than those shown using bold arrows. This implies that randomly populating a database will not necessarily make it more effective for solving all problems. One way of improving the efficiency of a synthesis process using a database, for solving a particular synthesis problem, would be to trim the database first by eliminating those building blocks from the database which cannot contribute to solving this problem, and then use this trimmed database in the synthesis process. For a given problem this trimming needs to be done only once, and then this trimmed database can be used as long as one wishes to synthesize solutions using various numbers of building blocks per solution, for this particular problem. However, if one wishes to use the original database for syn-

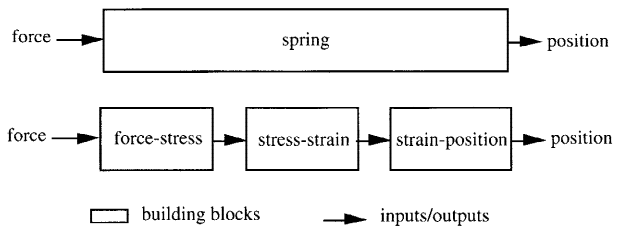


Fig. 5. The same transformation achieved by building blocks of different granularity.

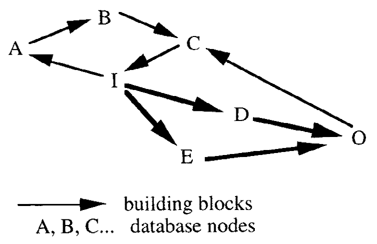


Fig. 6. An example database of building blocks.

thesizing solutions to a new problem, the original database needs to be trimmed for the new problem, after which this new, trimmed database can be used in as many synthesis experiments as required, by simply specifying the required solution size (i.e., number of building blocks per solution).

In this research, five such elimination rules have been developed, and the resulting simplified database is tested for its effectiveness by using it in generating solutions to a number of pilot problems, where the solutions generated are compared with those generated for the same problems using the original database. If the solutions generated using the simplified database contains all distinct solutions generated using the original database, and uses less time and memory, then this should demonstrate the potential of these simplifications in improving efficiency of synthesis without sacrificing effectiveness.

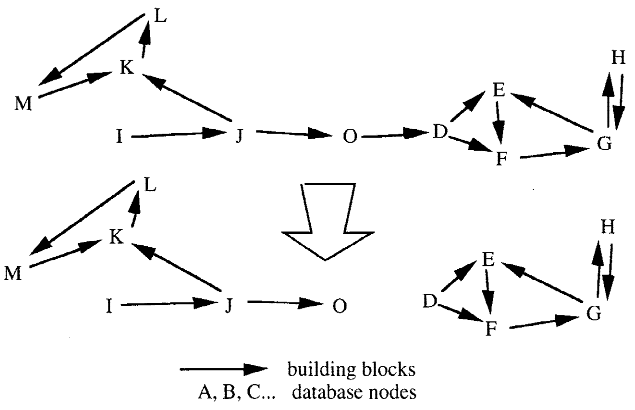


Fig. 8. Application of Rule 2 leads to elimination of link O → D.

6.2. Preprocessing rules

This section introduces and illustrates, using examples, the five rules that have been developed for preprocessing a database for a given problem.

Rule 1: Eliminate links directed towards the input node (as the task is to find ways of going from the input to the output, and this link will not help in that; see Fig. 7).

Rule 2: Eliminate links directed away from the output node (as the task is to find ways of going from the input to the output, and this link will not help in that; see Fig. 8).

Rule 3: Eliminate the links to each node (except the given input or required output) for which all links are either directed towards the node or away from it. In these nodes, it is possible either to reach the node or leave the node, but not both, and therefore impossible to go from one node to another (e.g., the input and output) via this node using these links (Fig. 9).

Rule 4: Eliminate each node (and their links) that is neither the given input nor the required output node, which has a single link towards it and a single link away from it, both of which are connected to the same node (as in these cases their use will lead to repetition of the same node, causing redundancy; see Fig. 10).

Rule 5: Eliminate all the nodes (and their links) that are neither directly nor indirectly connected to the input or output nodes, where an indirect connection between two nodes is defined as a connection between them via a set of links and nodes.

This rule is hard to implement without prohibitive computational expense, and if it could be implemented, this alone would be sufficient to prune the database, leaving only useful building blocks for synthesis. However, an approximation of this has been implemented that is relatively inexpensive and works always except for *cycles* containing both input and output nodes (see Appendix A). The approximation works like this. Suppose we wish to find those links to/from node A (in the case in Fig. 11 there are only four links: links 1 and 2 have A as their output, and links 3 and 4 have A as their input) that are not useful in the synthesis. These would be those through which it is not possible to go from the given input to the required output. Now, suppose

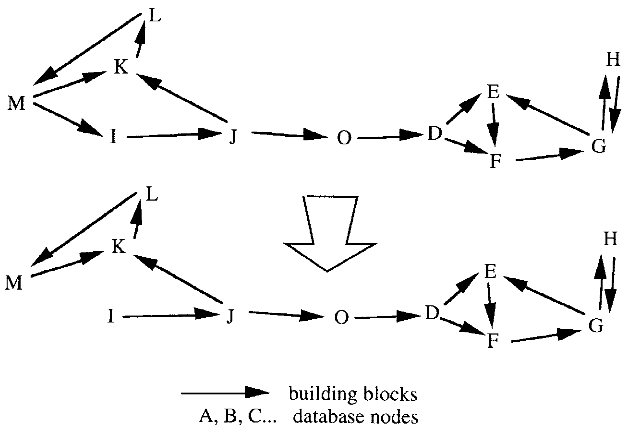


Fig. 7. Application of Rule 1 leads to elimination of link M → I.

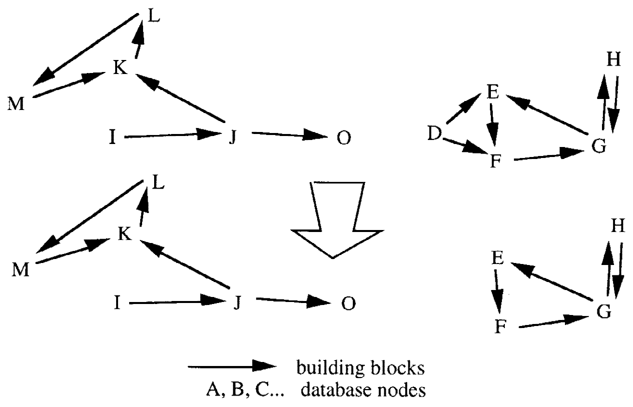


Fig. 9. Application of Rule 3 leads to elimination of links D → E and D → F.

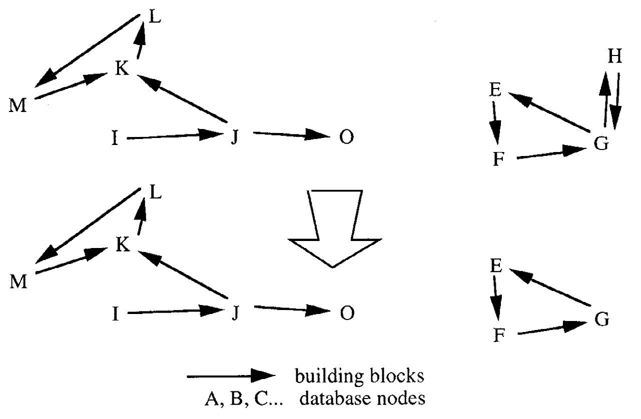


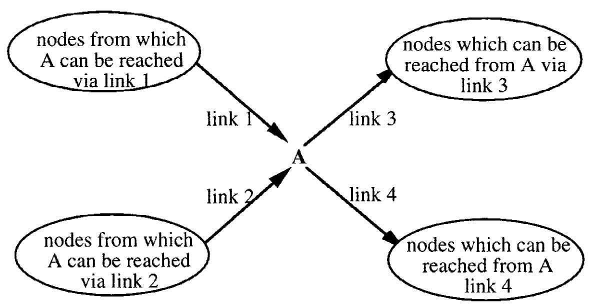
Fig. 10. Application of Rule 4 leads to elimination of links G → H and H → G.

we find all the nodes (we call them *back-sets*) that can be reached by going backwards from A via each distinct link from A; similarly, we find all the nodes that can be reached by going forward (*forward-sets*) from A via each distinct link from A. If any of these back-sets do not contain the given input node, then the link that connects this back-set to A will not be useful in synthesis, and can be safely eliminated. Similarly, none of the links connecting A to the forward-sets which do not contain the required output will be useful in synthesis, and can be safely eliminated. The process investigates the links associated with each node, and thereby eliminates isolated clusters of links and cycles that will not contribute to the synthesis. Repeated application of this rule, along with rules 1–4 should eliminate all the links that are not useful in solving a given problem (Fig. 12).

6.3. Preprocessing algorithm

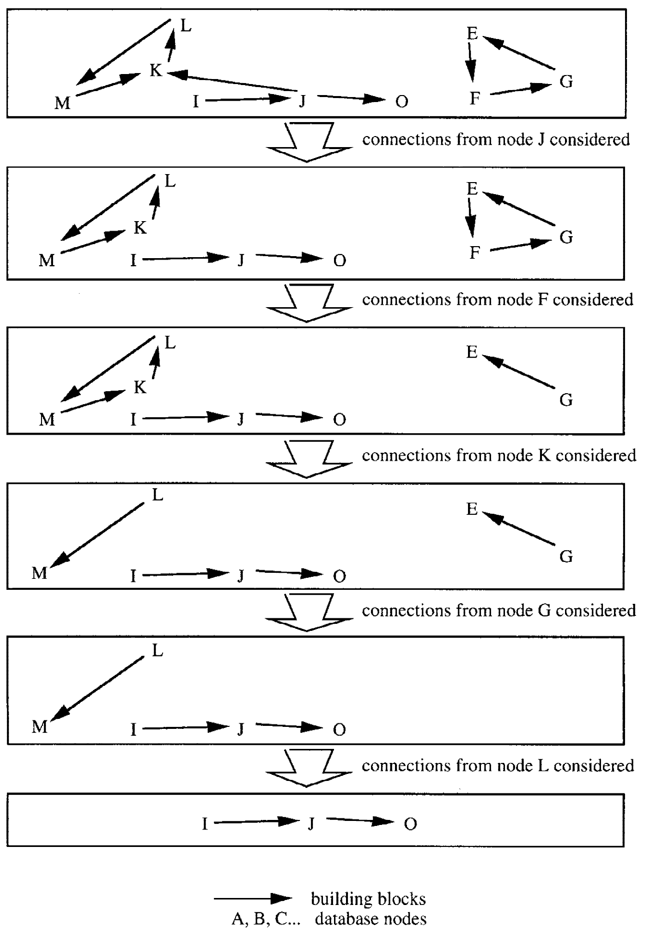
The algorithm contains three steps.

Step 1: By representing each building block in the database as a directed link between the input and output of the building block, and the input and output of the building block as two nodes, build a network of nodes and links to represent the database.



nodes from which A can be reached via link 1
link 1
nodes which can be reached from A via link 3
link 3
A
link 2
link 4
nodes from which A can be reached via link 2
nodes which can be reached from A link 4

Fig. 11. Implementing an approximate version of Rule 5.



L
K
M
I
J
O
E
G
F
connections from node J considered
connections from node F considered
connections from node K considered
connections from node G considered
connections from node L considered
building blocks
A, B, C... database nodes

Fig. 12. Application of Rule 5 leads to elimination of all links but I → J and J → O.

Step 2: Label the required input and output nodes as *input* and *output* nodes.

Step 3: Apply the five preprocessing rules, so as to simplify the network by eliminating some of its nodes and links. Each time the network is changed by the application of some rules, apply all the rules once again, until no further simplification of the network is possible.

7. ANALYSIS OF PERFORMANCE

7.1. Effect of preprocessing on computational performance of the synthesis procedure

The method for evaluation is to test the following, which together encapsulate the improvement expected of preprocessing the databases.

1. Resource for uni-dir. algorithm + preproc. DB < resource for uni-dir. algorithm + un-preproc. DB.
2. Resource for bi-dir. algorithm + un-preproc. DB < resource for uni-dir. algorithm + un-preproc. DB.

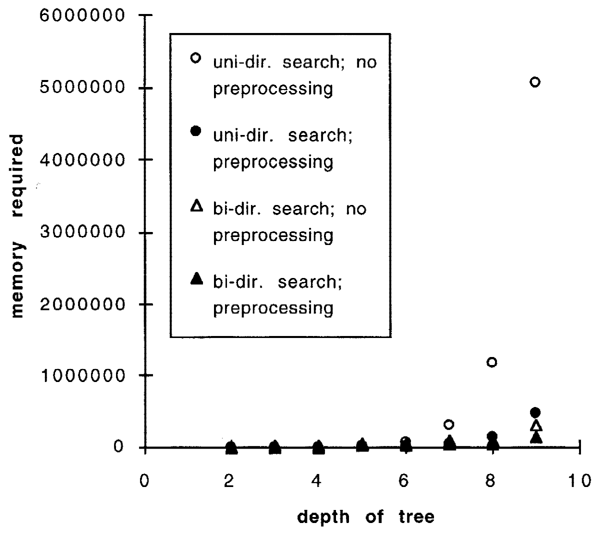


Fig. 13. Memory performance of procedure with and without database preprocessing.

3. Resource for bi-dir. algorithm + preproc. DB < resource for bi-dir. algorithm + un-preproc. DB.
4. Resource for bi-dir. algorithm + preproc. DB < resource for uni-dir. algorithm + preproc. DB.

7.2. Results

Figures 13 and 14 show the effect of database preprocessing, respectively, on memory and time required for generating the same set of solutions. In all figures in this section, memory is given in bytes and CPU time in seconds (the actual user time is between 4 and 20 times more due to garbage collection and other ancillary activities). The problem solved was acceleration sensing (as a transformation from acceleration to voltage). From Figure 13, it can be seen that a tenfold reduction in memory required is achieved (compare the top two plots) by preprocessing the database before using unidirectional search for synthesis, when the number of building blocks per solution synthesized is nine (given by *depth of tree* in the Fig. 13). Similarly, a twofold reduction in memory is achieved for the same solution size by preprocessing the database before using bidirectional search for synthesis (compare the bottom two plots in Fig. 13). Together, the introduction of bidirectional search instead of unidirectional search and of preprocessed database instead of randomly populated database achieved a 33-fold reduction in memory required. A similar trend has been observed in reduction in computation time (Fig. 14).

However, the performance of the preprocessing algorithm itself is combinatorial. As seen in Figures 15 and 16, preprocessing resource required increases faster than the increase in the size of databases processed. Each plot in these figures shows the preprocessing performance for a separate problem (e.g., acceleration, pressure, or strain sensing). However for a given design problem, a designer needs to preprocess the database only once before carrying out a series of synthesis experiments using the database. The larger this series (each experiment constitutes generating solutions having a given number of building blocks, which is a designer's choice), the less is the initial computational investment per experiment, and this gets increasingly justified as the allowable size of solutions (i.e., *depth of tree* in Figs. 13 and 14) increases. For instance, solving an acceleration sensing problem, using 9 building blocks per solution using an unpreprocessed database having 57 links requires 500,000 bytes of memory (Fig. 13), and preprocessing this database

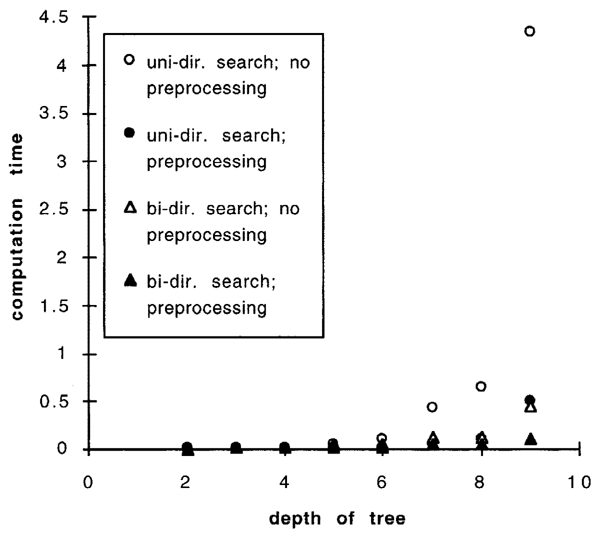


Fig. 14. Computational performance with and without database preprocessing.

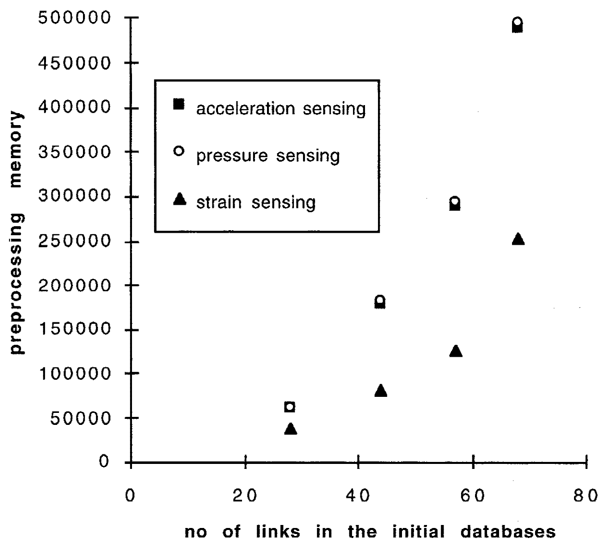


Fig. 15. Memory required in preprocessing with increase in size of database processed.

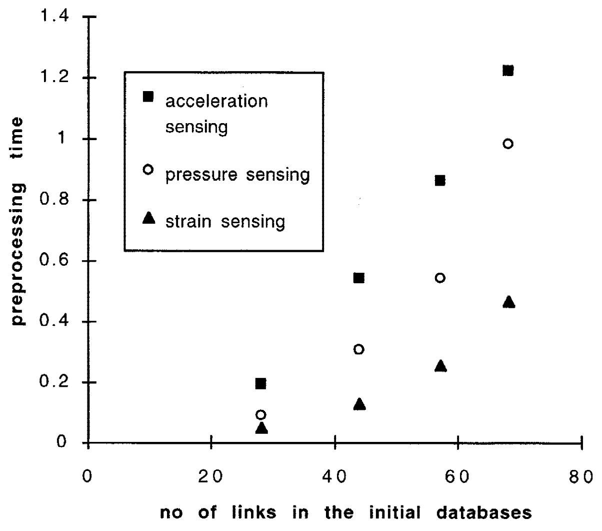


Fig. 16. Time required for preprocessing with increase in size of database processed.

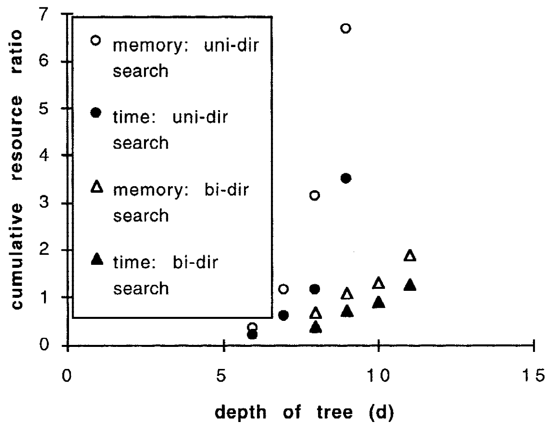


Fig. 17. Effect of preprocessing resources on performance on algorithms.

requires 300,000 bytes (Fig. 15). However, using the preprocessed database for bidirectional search requires about 100,000 bytes, making the combination still more profitable than using unpreprocessed bidirectional search.

To determine how an increase in solution size justifies the use of preprocessing, a parameter called *cumulative resource ratio* (CRR) is defined as:

$$\text{CRR} = \text{CRAUD}/(\text{CRAPD} + \text{RAPD}), \quad (1)$$

where CRAUD stands for *cumulative resource for algorithm using unpreprocessed database*. It is the total resource required for synthesizing solutions to a given problem of all sizes up to the maximum specified, using an unpreprocessed database. CRAPD stands for *cumulative resource for algorithm using preprocessed database*. It is the total resource required for synthesizing solutions for the same problem of all sizes up to the same maximum required size, using the database after it is preprocessed for this problem. RAPD stands for *resource for algorithm for preprocessing the database*, and is the resource required for preprocessing the database for this problem.

There can be two CRRs, one where the resource considered is memory required, and the other where it is the processing time. The value of a CRR shows how the resource required by an algorithm that uses an unpreprocessed database compares with the combined resource required by the same algorithm using a preprocessed database and by the algorithm for preprocessing the database. When CRR is 1, the overall resource required for preprocessing and using a preprocessed database breaks even with that using an unpreprocessed database. For any values of CRR larger than 1, preprocessing is more profitable for the size of solutions processed. Figure 17 shows how the CRRs for memory and time change as the maximum allowable size of solutions is increased for unidirectional and bidirectional search. The plots for unidirectional search have a faster rate of growth than bidirectional search. This means that preprocessing is more profitable with unidirectional search than with bidirectional search: the CRR goes up to 7 for a maximum solution size of 9 in unidirectional search whereas that for bidirectional search increases up to about 3. The rest of this section, therefore, investigates further how bidirectional search is affected by the preprocessing overhead.

Figure 18 shows how CRRs for memory and time for processing using bidirectional search change with the maximum size of solutions searched (i.e., maximum depth of tree searched) for two problems using a database having 57 links (i.e., building blocks) before preprocessing. Although there is considerable difference between the exact numbers in the two problem cases, the overall trend is similar: both CRRs reach the break even point at a moderate depth of 8–10, and grow up to 6 at a depth of 16, even for this small database.

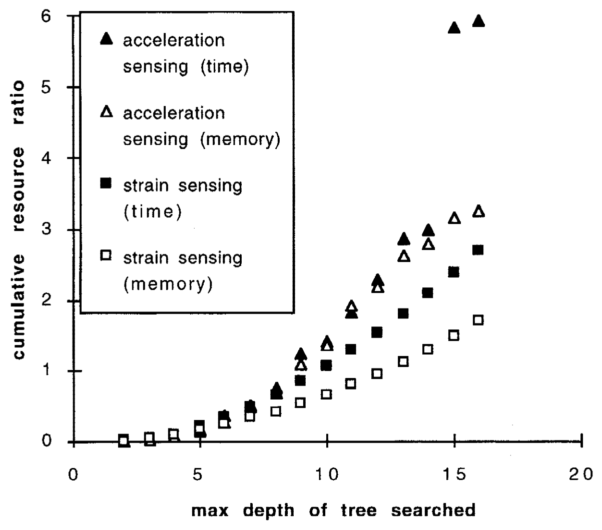


Fig. 18. Change in CRRs with maximum solution size in bidirectional search.

Figure 19 is a plot of the maximum size of solutions at which the CRR for memory breaks even against the size of the databases used for two problem cases using bidirectional search. Figure 20 is the corresponding plot for time as a resource. Though the exact value of the CRRs depends on the problem solved, the overall trend is similar: the maximum size of solutions at which resources break even decreases as the size of the database searched is increased.

Table 1 shows some of the data on which the figures cited above are based. The first column shows the intial size of some databases before preprocessing. The second column shows the percentage reduction they undergo as a result of preprocessing. The third and fourth columns show the maximum depth necessary for breaking even using these databases for time and memory, respectively, as the resource. Comparison of the results in the first two rows, for which the percentage reductions are high (47% and 46%, respectively), shows that the break-even depth is smaller for the larger database. The same is true for the databases in rows two and three. It appears that, for high values of percentage reduction, an increase in database size leads to a reduction in depth at the break-even point, making preprocessing increasingly cost effective. However, comparison of rows three and four shows that even though the database size is larger in row four, its percentage reduction is lower, almost halved, and the depth at the break even point does decrease with increase in database size. Similarly, if rows four and five are compared, it can be seen that, although the database size is larger in row five, its corresponding percentage reduction is lower. In this case, the break-even depth is either the same (for memory) or larger (for time) when the size of the database is larger. These two cases appear to illustrate that if percentage reduction is low, an increase in database size does not lead to a reduction in the depth required for breaking even. The general indication from empirical results is that, as long as the percentage reduction in the size of the database due to preprocessing is high, the larger the database preprocessed, the more cost effective preprocessing would be. However, a fuller analysis is not available at this stage of research.

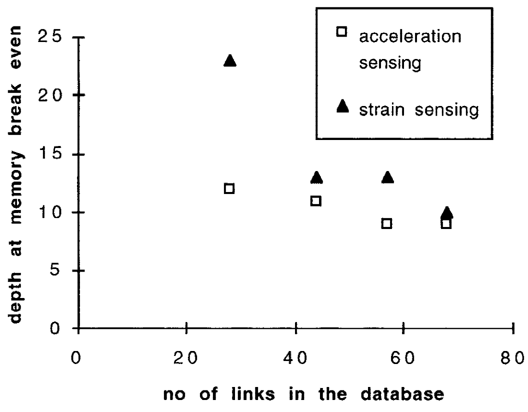


Fig. 19. Maximum size of solutions searched at memory Breaking Even *versus* database size.

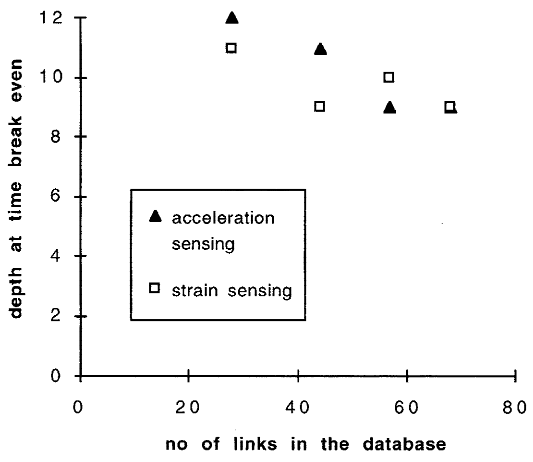


Fig. 20. Maximum size of solutions searched at Time Breaking Even *versus* database size.

8. SUMMARY, CONCLUSIONS AND FURTHER WORK

This article uses an engineering case study to identify the kinds of concepts generated by a computational compositional synthesis procedure, so as to develop ways of making the database of building blocks used in the synthesis process more effective and efficient. The case study reveals that, as well as generating concepts that are genuinely different from each other and therefore should be retained for consideration in the conceptual design phase, the procedure also generates concepts that are in many ways duplicated and need to be avoided. It is also found that only part of the database is useful in solving a given problem, and therefore the remaining part of the database need not be considered in the synthesis of concepts for this problem.

Efficiency and effectiveness are related. Avoiding the generation of uninteresting solutions leads to the saving of computational time, which amounts to generation of solutions

Table 1. *Database reduction in preprocessing and resulting change in break-even depth*

No. of links in DB before preprocessing	% reduction due to preprocessing	Depth at time breaking even	Depth at memory breaking even
28	46	12	12
44	47	11	11
57	38	9	9
68	22	9	9
90	15	10	9

of a higher quality using less resources than before. Considering only those building blocks that are useful in solving a given problem should also improve the efficiency of synthesis.

A number of rules for avoiding duplication of solutions are developed and used in populating and structuring databases of building blocks for use in computational-synthesis procedures. Rules for avoiding consideration of building blocks that are useless in solving a given problem are also developed, and are sewn together into an algorithm [implemented using Common-LISP on a LispWorks™ platform (Harlequin, 1991)] for preprocessing a database before using it to synthesize solutions for a given problem. Preprocessing provides a substantial savings of time and memory required for solving a given synthesis problem without sacrificing its effectiveness.

Being able to use large databases with a wide variety of building blocks is critical in enhancing the potential of compositional synthesis for generating innovative solutions, and many groups around the world are involved in developing such databases. However, without efficient synthesis procedures capable of handling these large databases, much of their potential for generating innovative solutions will remain unrealized. Structuring these databases using the rules mentioned in this paper and preprocessing them using algorithms reported here before searching them with improved search algorithms should enable effective and efficient generation of solutions.

However, the preprocessing algorithm is computationally expensive, although less so than synthesis using unpreprocessed databases, especially when the same preprocessed database is used in a series of computational experiments. It is found that, in general, as the size of the database to be preprocessed increases, preprocessing becomes increasingly profitable as long as it leads to a high percentage of reduction in database size. Further work will be focused on making the preprocessing algorithm more efficient and experimenting with larger databases to test the realistic effect of these algorithms.

ACKNOWLEDGMENTS

This project has been funded by the Engineering and Physical Sciences Research Council (EPSRC), U.K.

REFERENCES

Burgess, S., Moore, D., Edwards, K., Shibaike, N., Klaubert, H., & Chiang, H-S. (1995). Design application: The design of a novel microaccelerometer. *Workshop on Knowledge Sharing Environment for Creative Design of Higher Quality and Knowledge Intensiveness*. 12–13.

Chakrabarti, A. (1998). *A measure of the newness of a solution set generated using a database of building blocks and the database parameters which control its newness*. Technical Report No. CUED-C-EDC/TR64, Cambridge University, Cambridge, U.K.

Chakrabarti, A. (2001). Improving efficiency of procedures for compositional synthesis by using bi-directional search. *AI EDAM* (in press).

Chakrabarti, A., & Bligh, T.P. (1994). Functional synthesis of solution-concepts in mechanical conceptual design. Part I: Knowledge representation. *Research in Engineering Design 6(3)*, 127–141.

Chakrabarti, A., & Bligh, T.P. (1996*a*). Functional synthesis of solution-concepts in mechanical conceptual design. Part II: Kind synthesis. *Research in Engineering Design 8(1)*, 52–62.

Chakrabarti, A., & Bligh, T.P. (1996*b*). Functional synthesis of solution-concepts in mechanical conceptual design. Part III: Spatial configuration. *Research in Engineering Design 8(2)*, 116–124.

Chakrabarti, A., & Bligh, T.P. (1996*c*). An approach to functional synthesis of design concepts: Theory, application, and emerging research issues. *AI in Engineering Design, Analysis and Manufacturing 10(4)*, 313–331.

Chakrabarti, A., Johnson, A.L., & Kiriyama, T. (1997). An approach to automated synthesis of solution principles for micro-sensor designs. *Proc. International Conference on Engineering Design ICED'97*, 125–128.

Finger, S., & Rinderle, J.R. (1990). *A transformational approach for mechanical design using a bond graph grammar*. EDRC Report no. 24-23-90. Carnegie-Mellon University, Pittsburgh, PA.

Harlequin. (1991). *LispWorks: The Reference Manual*, Harlequin PLC, U.K.

Hoeltzel, D.A., & Chieng, W-H. (1990). Knowledge-based approaches for the creative synthesis of mechanisms. *Computer-Aided Design 22(1)*, 57–67.

Hoover, S.P., & Rinderle, J.R. (1989). A synthesis strategy for mechanical devices. *Research in Engineering Design 1(2)*, 87–103.

Ishii, M., Tomiyama, T., & Yoshikawa, H. (1994). A synthetic reasoning method for conceptual design. In *Towards World Class Manufacturing*, (Wozny, M., & Olling, G., Eds.), (pp. 3–16). Elsevier Science, North-Holland, Amsterdam.

Khang, J.H.L. (1998). Embodiment modelling with parameter trees. Ph.D. Thesis. University of Cambridge, Cambridge, U.K.

Kota, S., & Chiou, S.-J. (1992). Conceptual design of mechanisms based on computational synthesis and simulation. *Research in Engineering Design 4*, 75–87.

Malmqvist, J. (1993). Computer-aided conceptual design of energy-transforming technical systems. *Proc. Int. Conf. on Engineering Design, ICED'93*, 1541–1550.

Pahl, G., & Beitz, W. (1984). *Engineering Design: A Systematic Approach*. Springer-Verlag, New York.

Paynter, H.M. (1961). *Analysis and Design Of Engineering Systems*. The MIT Press, Cambridge, MA.

Prabhu, D.R., & Taylor, D.L. (1988). Some issues in the generation of the topology of systems with constant power-flow input-output requirements. *Proc. ASME Design Automation Conference*, 41–48.

Roth, K. (1970). Systematik der Machinen und ihrer Mechanischen Elementaren Funktionen. *Feinwerktechnik 74*, 453–460.

Selutsky, A.B. (Ed) (1987). *Daring Formulae of Creativity*. Karelia, Petrozavodsk, Russia (in Russian).

Sushkov, V., Alberts, L., & Mars, N.J.I. (1996). Innovative Design Based on Sharable Physical Knowledge. In *Artificial Intelligence in Design '96*, (Gero, J.S., & Sudweeks F., Eds.), pp. 723–742. Kluwer Academic, Dordrecht, The Netherlands.

Taura, T., Koyama, T., & Kawaguchi, T. (1996). Research on natural law database. *Joint Conf. Knowledge Based Software Engineering '96*. Sozopol, Bulgaria.

Tsourikov, V.M. (1995). Inventive machine: 2nd generation. *AI and Society 7(1)*, 62–78.

Ulrich, K.T., & Seering, W.P. (1989). Synthesis of schematic descriptions in mechanical design. *Research in Engineering Design 1(1)*, 3–18.

Umeda, Y., & Tomiyama, T. (1997). Functional reasoning in design. *IEEE Expert: Intelligent Systems and Their Applications 12(2)*, 42–48.

Welch, R.V., & Dixon, J.R. (1994). Guiding conceptual design through behavioral reasoning. *Research in Engineering Design 6(3)*, 169–188.

APPENDIX A: A DATABASE BEFORE AND AFTER PREPROCESSING

The following figures show all the links in one of the databases preprocessed (the second data point from the left in Figs. 15 and 16). This database has 44 links, and those shown in bold arrows in Fig. A1 are deleted by the preprocessing

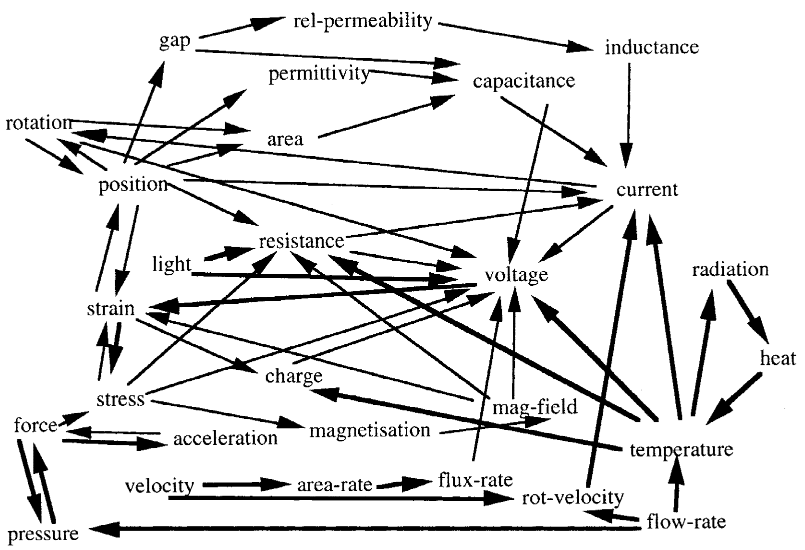


Fig. A1. The database before preprocessing has 44 links.

algorithm for an acceleration sensing problem defined as that of finding routes from the acceleration node to the voltage node in this database. Notice that light → resistance and light → voltage links are eliminated because the light node does not have any input link to it, which means there is no way of getting to this node from any other node. For a similar reason, all links from flow-rate and from velocity node can be eliminated. Once these are eliminated, area-rate and rot-velocity also become nodes without input links, and can be eliminated, which, in turn, makes all links from flux-rate node eligible for elimination for the same reason. The voltage → strain link is eliminated because voltage cannot be reached using this link. Similarly, force → acceleration link is of no use because acceleration cannot be left using this link. As the only route from acceleration is via force, and as the only route from pressure is via force, the pressure → force and force → pressure links would be of no use in solving the problem of going from acceleration to force, and can be eliminated.

Then, as the only route from acceleration is first through force and then through stress, the link connecting strain → stress is of no use, as it forces visiting the same node more than once. The links among temperature, heat, and radiation form a cycle, using any of which would require visiting the temperature node more than once during a move from acceleration to voltage via temperature, and should be eliminated. This leaves temperature as a node with output links only, and can also be eliminated. This completes the process of preprocessing (Fig. A2).

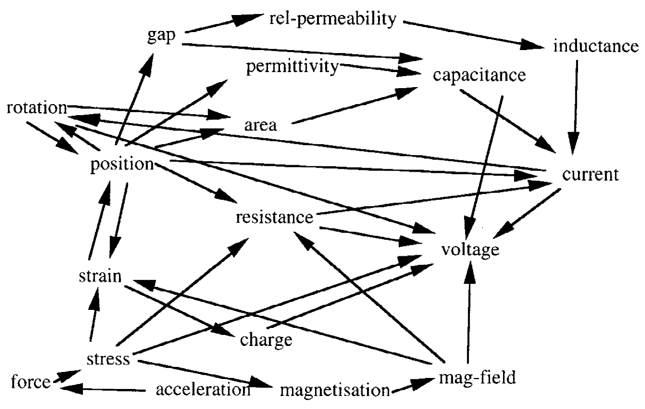


Fig. A2. The same database after preprocessing has 35 links.

APPENDIX B: WHY CYCLES CANNOT BE DETECTED BY RULE 5

Figure B1 shows a cycle of links in some database, that is, a chain of links that has the same node as its output and input. This specific cycle contains those nodes I and O, which are the required input and output of a given problem. In this case, we need to check each link in this cycle to see if it is useful for going from the input to the output. The way we do this using Rule 5 is by going backward from each node to see whether the required input node can be reached without repeat visit to any node. The link connecting the node under consideration that initiated the backward movement should, in this case, be considered useful. Similarly, going forward from the node considered to check whether the output node can be reached without revisiting any node in this

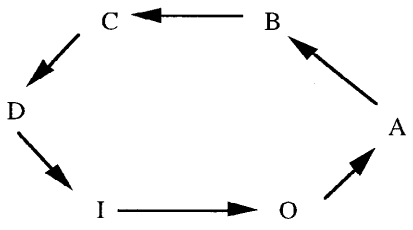

$$\begin{tikzcd} & C \arrow[dl] & B \arrow[l] & \\ D \arrow[dr] & & & A \arrow[ul] \\ & I \arrow[r] & O \arrow[ur] & \end{tikzcd}$$

Fig. B1. A cycle of links containing the input and the output node.

process should allow us to check the usefulness of the forward link of the node considered through which such forward movement was initiated. Scrutinizing the cycle in Figure B1 reveals that no link in this figure, except I $\rightarrow$ O, is useful for transforming I to O. However, starting from any of their nodes, I and O can be reached by going backwards and forwards, respectively, that is, by applying Rule 5.

Amaresh Chakrabarti received a B.E. in Mechanical Engineering from the University of Calcutta in 1985, an M.E. in Mechanical Design from Indian Institute of Science in 1987, and a Ph.D. in Engineering Design from Cambridge University in 1991. He has since been associated with the Engineering Design Centre at Cambridge University, as a Senior Research Associate. In 1994, his software FuncSION won a prize in the UK Morgans-Grampian Manufacturing Industry Achievements Awards competition. His main interest is in design methodology, particularly in the earlier phases of design. This includes requirements identification, functional representation, conceptual design, and engineering design research methodology.