Artificial Intelligence for Engineering Design, Analysis and Manufacturing (2001), *15*, 233–249. Printed in the USA.
Copyright © 2001 Cambridge University Press 0890-0604/01 $12.50

Design synthesis knowledge and inductive machine learning

S. POTTER, M.J. DARLINGTON, S.J. CULLEY,* AND P.K. CHAWDHRY
Engineering Design Centre, Department of Mechanical Engineering, University of Bath, Bath, BA2 7AY, U.K.

(RECEIVED May 15, 1999; ACCEPTED November 1, 2000)

Abstract

A crucial early stage in the engineering design process is the conceptual design phase, during which an initial solution design is generated. The quality of this initial design has a great bearing on the quality and success of the produced artefact. Typically, the knowledge required to perform this task is only acquired through many years of experience, and so is often at a premium. This has led to a number of attempts to automate this phase using intelligent computer systems. However, the knowledge of how to generate designs has proved difficult to acquire directly from human experts, and as a result, is often unsatisfactory in these systems. The application of inductive machine learning techniques to the acquisition of this sort of knowledge has been advocated as one approach to overcoming the difficulties surrounding its capture. Rather than acquiring the knowledge from human experts, the knowledge would be inferred automatically from a set of examples of the design process. This paper describes the authors' investigations into the general viability of this approach in the context of one particular conceptual design task, that of the design of fluid power circuits. The analysis of a series of experiments highlights a number of issues that would seem to arise regardless of the working domain or particular machine learning algorithm used. These issues, presented and discussed here, cast serious doubts upon the practicality of such an approach to knowledge acquisition, given the current state of the art.

Keywords: Conceptual Design; Design Synthesis; Inductive Machine Learning; Heuristic Knowledge

1. INTRODUCTION

Engineering design may be seen as the task of generating the description of some artefact that meets a given specification which expresses the functional and performance requirements for the artefact, along with any constraints within which it must operate. A crucial early stage in the design process is the *conceptual design* phase (Pahl & Beitz, 1996), during which an initial solution to the design problem is devised. This initial solution will incorporate physical solutions to the essential functional or performance features of the problem. This is the phase of the design process that "makes the greatest demands on the designer, and where there is most scope for striking improvements . . . and where the most important decisions are taken" (French, 1985).

At the heart of the conceptual design phase lies the *synthesis* of potential solutions by the designer. When discussing the synthesis of solutions, design textbooks (e.g., Pahl & Beitz, 1996; Ullman, 1997) suggest a number of approaches, ranging from searching for analogous solutions in other domains to convening group brainstorming sessions. However, while they might provide a stimulus to designers, or situations conducive to the development of ideas, none of these suggested approaches can *guarantee* a valid solution as an outcome: Ultimately the synthesis of solutions is dependent on the knowledge of the designer, and this knowledge cannot easily be taught or passed on (Reich & Fenves, 1992). This is due, at least in part, to the *heuristic* nature of synthesis knowledge—it is acquired through the designer's experience of the task and of the domain—and the length of time and the amount of effort that is needed to accumulate this experience often ensures that this knowledge is in short supply.

The advent of computers, and the ease with which their programs may be duplicated and transferred, along with

*Address correspondence and reprint requests to: S.J. Culley, Engineering Design Centre, Department of Mechanical Engineering, University of Bath, Bath, BA2 7AY, U.K. E-mail: S.J.Culley@bath.ac.uk

https://doi.org/10.1017/S0890060401153047 Published online by Cambridge University Press

the interest generated by research into artificial intelligence (AI), has given rise to the desire to construct computer systems which incorporate design knowledge and which can apply it to solving new design problems. With the scarcity of synthesis knowledge for conceptual design, and the importance—and difficulty—of this phase, the automation of the conceptual design task would be of great benefit in many domains. While initial attempts at producing intelligent design systems [e.g., that of McDermott (1982)] were promising, it was quickly recognized that a major difficulty in constructing such systems lay in the acquisition and expression of the synthesis knowledge. Conventional knowledge-acquisition techniques, by which "knowledge engineers" aim to capture knowledge directly from human experts, have been found lacking when attempting to acquire experience-based heuristic knowledge (Berry, 1987; Hart, 1988; Gillies, 1996). As a result, a number of authors (e.g., Michie, 1982; Lenat, 1983; Reich & Fenves, 1989) have suggested that at least some of this knowledge might be acquired automatically through the use of Machine Learning (ML).

Much effort has been devoted over the last few decades to the development of *inductive* ML algorithms, that is, algorithms that are able to acquire generalized knowledge by induction over a given set of examples of some event. In design contexts, the idea of acquiring the synthesis knowledge from examples of previous design episodes is an appealing one. Accordingly, this paper is the result of an investigation into the suitability of available inductive ML algorithms for learning conceptual design synthesis knowledge. The paper will describe some of the issues that have arisen which have hampered the learning effort and restricted the quality of the acquired knowledge—and which, the authors believe, will arise during any attempt to acquire heuristic knowledge in this manner using current techniques.

1.1. Content of this paper

The following section describes some of the previous approaches to capturing design synthesis knowledge using inductive ML. The realization that this previous work has been limited in scope and has left unanswered the more general questions about the potential of the technique was to give the research described in the remainder of the paper an extra impetus.

For the purposes of this research, a specific design task would be needed to provide a working domain. That chosen was the conceptual design of fluid power circuits, a task that was felt to display many of the essential features common to conceptual design. This task is described in the third section of the paper, along with a manner of representing the task in computer systems and the general issues surrounding the development of such a representation. The basis of the inductive learning about this task would be a set of examples of this design task; the fourth section relates the problems that are encountered when constructing such an archive.

Subsequent sections describe the general inductive ML approach, and the steps involved in constructing three particular design systems using machine-learned knowledge. Each of these systems uses one of a different class of ML algorithm. In the course of building these, and other design systems, a number of general issues have arisen; the final sections of the paper describe these issues and discuss their wider implications for the inductive learning approach.

2. A REVIEW OF THE INDUCTIVE ACQUISITION OF DESIGN SYNTHESIS KNOWLEDGE

ML techniques have been applied to a number of design learning tasks, ranging from the acquisition of analytical knowledge to learning about the design process itself. This work has been substantial enough to warrant several special issues of this journal (Maher et al., 1994; Duffy et al., 1996, 1998), and Duffy (1997) provides a useful overview of research into the application of ML techniques in design.

Amongst this research there has been a number of examples of the use of inductive ML to acquire design synthesis heuristics, the topic with which this paper is principally concerned. A theme common to a number of these approaches is that of *conceptual clustering*. Based upon recognized similarities within the descriptions of events (which correspond to design cases), these techniques construct a model of the domain represented by these events, usually in the form of a hierarchical structure of "concepts", becoming gradually more specific to individual events as the structure is traversed. Reich and Fenves (1992, 1995) use a clustering algorithm to create such a conceptual hierarchy representing the descriptions of bridge designs. Based on its design requirements, a new problem can be matched to the most similar cluster in the hierarchy, and the solutions to the examples stored under this cluster used as the basis for a solution to this new problem. Other researchers who have used clustering approaches include Duffy and Duffy (1996), Lu and Chen (1987), who go on to use their clustered data to inductively produce predictive rules for class membership, and Maher and Li (1994) who include a learning component that attempts to learn additional domain knowledge to assist future design. A later section of this paper discusses a clustering approach adopted by the authors of the present paper. This work has highlighted a number of problems with such approaches.

Aside from clustering approaches, Ivezic and Garrett (1994) use an *artificial neural network* to try to predict the values of form and behavior attributes of computer systems, whereas Coyne et al. (1993) use a network to try to complete partial designs of rooms, in terms of their contents. Artificial neural networks and their application to design problems will also be discussed in a later section of this paper.

https://doi.org/10.1017/S0890060401153047 Published online by Cambridge University Press

Notwithstanding the promising results reported by these authors, there has been relatively little research into the automatic acquisition of synthesis knowledge, especially when one considers the value (and scarceness) of this knowledge. This may well be due to the particular difficulties that accompany this approach. In the following sections, through a discussion of the authors' experiments, these difficulties will be described and discussed.

3. DESCRIPTION OF THE DESIGN PROBLEM

The task in question is the conceptual design of fluid power circuits. These circuits convert rotational mechanical energy into fluid energy (*via* a pump), which is then transmitted by connecting pipes to some remote point of actuation, at which it is transformed back into mechanical energy. In addition to pumps and actuators, and the connecting pipe work, there are a number of other standard domain components which serve to control the pressure or flow rate of the fluid, or to direct its flow. Figure 1 shows a schematic of a simple fluid power circuit.

At the conceptual design stage, then, the task is to translate a customer-supplied design specification, which describes the desired functionality of the actuator in abstract terms, into a configured circuit that is able to supply this functionality. The components in this circuit will be described using generic terms; later stages in the design process (with which the work reported here is not concerned) will involve selecting specific manufacturers' models for these, and providing values for associated operational parameters.

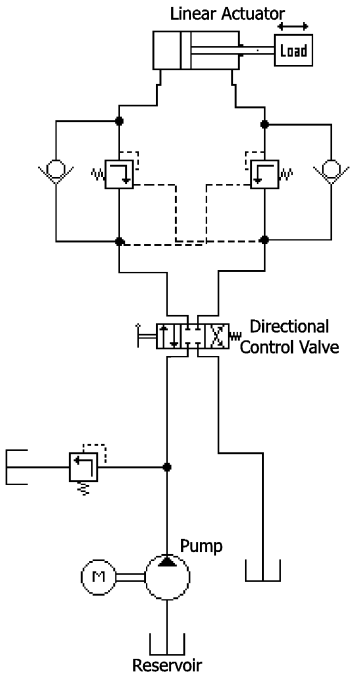


Fig. 1. An example fluid power circuit.

More generally, this task is typical of a common type of conceptual design problem, that of *configuration design* (Mittal & Frayman, 1989): Some description of a system must be synthesized using available domain components to meet a customer-supplied set of functional requirements.

So, for this design task, the conceptual design synthesis heuristics permit a translation of the set of functional requirements into the description of an appropriate circuit. Accordingly, for the purposes of learning this knowledge inductively from a set of design examples, each example is defined by a set of functional requirements, and the corresponding circuit solution. As a preliminary to the application of ML, formal methods of representing both requirements and circuits must be devised, and a set of examples collected and described according to these representations.

3.1. The representation of the design problem

As has often been noted [e.g., by Winston (1993)], when constructing intelligent systems, a good representation of the problem is crucial, and this case is no exception. Here, each design example must be represented in formal, computationally tractable terms; in practice, this means that sets of parameters for describing the functional requirements and the circuit solutions, along with the values that each of these parameters can assume, must be identified.

Unfortunately, producing this formal description of the problem is not a trivial task, and requires a certain amount of knowledge about the task. For simple problems, the inputs and outputs of the process are immediately obvious; however, this is not the case for the majority of conceptual design tasks. Since the task is at the functional level, the elements comprising the descriptions must be represented accordingly. This may mean, for example, that it is more appropriate to consider a group of components that together achieve some atomic function as constituting a single solution parameter rather than a number of separate parameters, each corresponding to a single component. The need to make representational choices of this sort raises the question of the source and availability of the knowledge required to do so—this is quite detailed knowledge of the design task in question.

Ideally, the form and content of the devised representations should be influenced solely by the intention to reflect accurately the essentials of the task itself, and not by the intention to use any particular method to implement it on computer. However, in this case, given the scarcity of synthesis knowledge and the desire to investigate the possibilities of using ML, it is almost inevitable that the representations are also influenced by the manner in which the ML algorithms expect data to be presented. The effect of this can be representations that are less faithful to the task than they might otherwise be. Since the task of learning this design knowledge may be viewed as one of recognizing consistent relationships amongst the elements describing the problem, this

https://doi.org/10.1017/S0890060401153047 Published online by Cambridge University Press

is only possible if these elements are themselves described consistently. As a consequence, the algorithms generally demand rigorous consistency in the manner in which examples are described. Typically, it is expected that examples are represented in the form of a set of values, one for each of a fixed number of parameters.

Hence, there are a number of (not necessarily compatible) considerations that inform and dictate the representational decisions made. To illustrate this more practically, the representation of the design requirements and of the design solutions for the fluid power circuit design task will now be discussed.

3.1.1. Requirements representation

The expression of design requirements, in particular, is fraught with difficulties in most domains. The scope for potential requirements, both in terms of their content and the manner in which they are expressed, is extremely wide. In general, all these difficulties arise because design is fundamentally a human process, and takes place in a human environment. As with all communication, expression of the requirements takes place in this context, and assumes shared linguistic, cultural, and social references between the customer and the designer. In addition, the requirements are often developed over time, as a result of this interaction between customer and designer. Acquiring the vocabulary and semantics necessary to discuss domain problems is itself a complex learning task.

Obviously, these points of reference are unavailable to a computer. When attempting to model this on computer, an attempt must be made to compensate for this lack by introducing some representation that allows the communication of sufficient information to produce the design, and which permits a realistic breadth of expression, but which is also tightly constrained, in that it defines the totality of the common language between user and computer, preventing the introduction of *ad hoc* terms, which, by definition, would be outside the "understanding" of the computer.

It should go without saying, then, that the description of the requirements needs to refer to those elements of the functionality that have a bearing on the final conceptual design solution, and furthermore, that the values that these elements can assume form a useful (in terms of the task in hand) partitioning of the requirements "space".

To produce such a formalism, there are two practical approaches available: consultation of an expert designer to try to access directly a suitable representation, or else, familiarization with the chosen domain and task on the part of the knowledge engineer. Both approaches introduce their own problems. As remarked earlier, direct knowledge acquisition from experts is difficult and problematic. On the other hand, task familiarization requires the knowledge engineer to learn about the domain, which can be an even lengthier process, and, if not done thoroughly, can lead to the introduction of inappropriate representations. The approach chosen here was that of familiarization with the task. This was done partly because a suitable design expert was unavailable, and partly as it was felt that such a familiarization would be useful for understanding and modelling the entire problem.

The resulting formalism developed for representing the requirements consists of a set of 14 parameters; to describe one particular design problem, an appropriate value must be associated with each parameter. Principally, these parameters describe the functionality that any solution must provide. For example, one parameter describes whether or not the speed of the actuator motion should be independent of the magnitude of the load—this can assume the value "yes" or else "no" as desired. In addition to these functional parameters, there are some additional parameters describing the context within which the functionality is to be provided—for example, the maximum magnitude of the load that is to be moved. This is nominally a real-valued parameter, but it was decided that this, and the other "context" parameters, could be expressed through the use of "high", "medium", and "low" values corresponding to appropriate numerical ranges. It was felt that this representation would impart the information necessary for the design task (and, in so doing, potentially facilitate the learning by making regularities within the data explicit).

So it can be seen that a particular set of requirements is described by 14 symbolic values. This representation was felt to allow the communication of sufficient information about the desired circuit to permit its conceptual design.

3.1.2. Design solution representation

On paper at least, the representation of solutions would seem to present a simpler task in this case, since circuits consist of a number of explicit, predefined components. An examination of sample circuits led to the recognition that the majority of design solutions share a basic framework—the solution *template* [Figure 2(a)]. The template embodies the functionality integral to all solutions (it includes a power source in the form of a basic pump, an actuator, and the facility to direct flow from the pump to the actuator). Moreover, the template has a number of labelled slots, into which additional components can be placed in order to provide the precise functionality demanded by the requirements. Hence, by reference to these slots, the template also provides a convenient context for describing complete solutions: A solution consists of the template along with the set of additional components, their corresponding slot positions, and orientations within the slot. Any slots without components are considered to be filled by a direct pipe connection. (Note that although not *all* circuit solutions conform to this template, it was considered to be prevalent enough to be adopted as a useful representational expedient.)

Although, in theory, any component in this domain could be placed in any slot, in practice only certain components used in certain ways are found to provide useful functionality (with, sometimes, several components being necessary to supply a single function). The next step in describing

https://doi.org/10.1017/S0890060401153047 Published online by Cambridge University Press

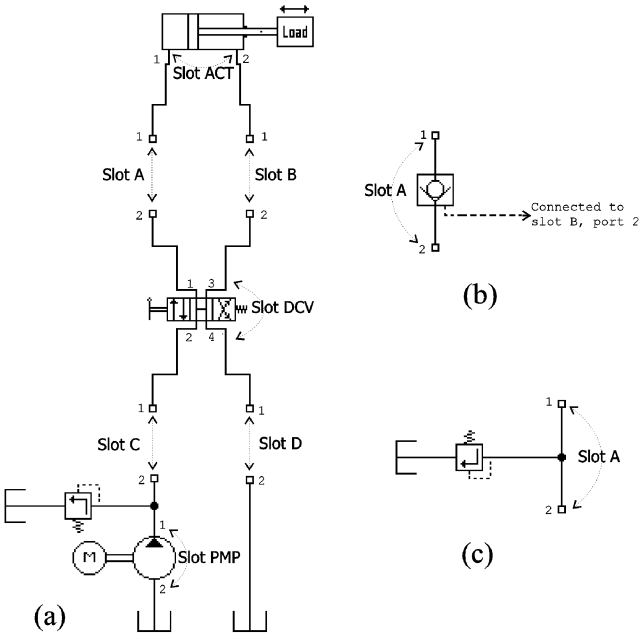


Fig. 2. (a) The solution template; (b) solution element *POCV_A*—a *pilot-operated check valve* in slot A; and (c) solution element *PRV_A*—a *pressure relief valve* in slot A.

solutions, then, is to identify this set of useful components and the positions and orientations in which they are used. The combination of component, slot, and orientation defines a solution *element*: Through inspection of the available design cases, 17 such elements have been defined. Figures 2(b) and (c) show two of the identified solution elements. Since each element can usefully appear only once in a solution, and given the constant template, a particular solution can be described by labelling each element as either present or absent, as appropriate. Hence, 17 parameters, each assuming the value "present" or else "absent" represent design solutions. Once again, this representation includes task knowledge of a rather detailed nature.

3.2. Description of the design problem—issues

A number of issues are raised when considering the representation of design problems in ML contexts:

- Appropriate representations are crucial if the task is to be learned about and then performed adequately. However, some compromise must usually be found between producing the "best" representation and producing a representation that can be used with ML algorithms.
- Useful representations often incorporate much knowledge about the task in hand in the manner in which the problem is described. Hence, the definition of representations is itself a knowledge-acquisition task. Currently there are two practical approaches available, namely, some interaction with a human design expert to access his/her representations, or some degree of task familiarization on the part of the knowledge engineer. Each approach introduces its own difficulties.
- The representation of requirements can be particularly difficult, since it involves the restriction of theoretically unconstrained expressiveness into a limited set of terms. Any representation can only be an approximation to the actual description: The goal must be to ensure that it contains those elements most influential in the design process.

4. THE SOURCE OF HEURISTICS—A FLUID POWER DESIGN ARCHIVE

The hypothesis, then, is that the conceptual design synthesis knowledge applied by a designer is implicit in the combination of requirements specification and corresponding solution, and that this knowledge can be extracted through the application of inductive ML. To this end, an *archive* of this design information has been collected: Each member of this archive consists of a design requirement and the corresponding solution, represented in the form described above. This archive is to form the basis of the training data supplied to ML algorithms.

The examples in this archive come from a number of sources: Some are from the design archives of companies, some from training material, and others from textbooks. Whatever their origin, the assumption is made that each example is of a *successful* design case.

4.1. The archive of design examples—issues

The construction of the archive, the nature of its content, and its intended use, raise a number of significant issues. These will now be discussed.

4.1.1. The number of examples

Presumably, the synthesis knowledge for this design task is complex, and so, for it to be learned, it is essential that the archive is fully representative of this knowledge. Hence, there would seem to be a need for a large archive. However, readily available examples, sufficiently well documented for incorporation into the archive, were found to be few in number (the archive consists of 30 examples). It seems that organizations in this field tend to specialize in a particular area, and so do not possess a wide range of sample designs, and, moreover, that restrictions are often placed on the release of design information. It would seem necessary, then, that any approach to the automatic acquisition of heuristic knowledge from examples will have to work within these constraints.

4.1.2. The description of the examples

The examples must be redescribed to conform to the adopted formal representations. This involves some interpretation of the meaning of the design, particularly in the

https://doi.org/10.1017/S0890060401153047 Published online by Cambridge University Press

case of the requirements; misinterpretation can introduce errors into the archive, and information can be lost (or added).

4.1.3. The incompleteness of the examples

Many examples are incompletely recorded. Chiefly, the problem lies in the description of the requirements, as solutions tend to be well described (of necessity, since they have to be physically constructed at some stage). In addition to poor documentation practices, this incompleteness might be due to assumptions made by designers about what is "obvious", and can thus be left unsaid.

Some ML algorithms are capable of dealing with missing values of this sort, typically by assuming the most likely value. However, here the decision was made to try to increase the completeness of the data prior to learning, by "reverse engineering" missing requirements values. Based on the solution description, assumptions are made about the functionality of the system, and, hence, about the content of the original requirements. Again, there is always the danger that such an approach will introduce errors into the archive.

4.1.4. The quality of the examples

Each solution in the archive is assumed to be an example of a "good" design. However, this notion of goodness is ill-defined. For example, different companies may apply different criteria when judging if a solution meets the stated requirements. In addition, circuits operate over some timescale. What may initially seem to be a good solution may fail after a number of years' service due to flaws in the initial design. Furthermore, whenever dealing with design documentation, some allowance must be made for the possibility of it containing incorrectly recorded or otherwise corrupted values.

4.1.5. The expression of the examples

For the suggested inductive ML approach to work, the data must exemplify consistent "causal" relationships between requirements and solutions, in the sense that (when the solution is physically constructed) particular elements in the solution should consistently "cause" particular elements of the requirements to be satisfied. However, this causal relationship might appear to be broken in an example if, say, a solution actually provides more functionality than is explicitly demanded in the set of requirements. While this may not represent a "wrong" solution (if the additional functionality does not interfere with the primary purpose of the circuit, and the customer incurs no extra cost), it does present an obstacle to the automatic acquisition of these relationships. Again, the approach adopted here has been to try to reverse engineer the requirements so that they reflect the total functionality embodied in the solution.

4.1.6. The style of design solutions

A further issue that crops up when attempting to amass an archive for learning purposes is that of the *style* of solutions. This problem is most apparent when two or more different design solutions appear to match the same requirements. This may be because the devised representation of the requirements is not sufficiently expressive to capture the distinction that has led to the different designs. Alternatively, it may simply be a reflection of the fact that, in complex domains, it is possible to achieve the same effect in two or more different ways. Whatever the reason, it poses a problem for inductive ML in that there appears to be a contradiction in the examples: A design problem is associated with two or more solutions—which association is to be learned?

A further stylistic problem arises due to the fact that the example designs collected in the archive have been produced over a number of years. Over time, the nature of design problems can and does alter: New components become available, legislation is introduced, working practices change, and so on, all of which may alter the nature of the solutions that are produced. This "evolution" of the domain may be reflected in an apparent lack of consistency or regularity in designs.

Even assuming that obviously contradictory examples are purged from the archive, this problem may manifest itself more subtly in the form of very different solutions to similar (but not identical) sets of requirements. Such phenomena in the training data can restrict the ability of a ML algorithm to generalize over the data.

5. MACHINE-LEARNING DESIGN SYNTHESIS KNOWLEDGE: EXPERIMENTAL METHOD

During conceptual design, the description of a solution is synthesized in response to some set of requirements. It seems clear that the knowledge required to perform this task describes in some form the relationship that exists between elements of the requirements and elements of the solution. The learning task, as considered here, is one of identifying these relationships, which correspond to the synthesis heuristics. It is hypothesized that these are implicit in examples of previous design episodes (each consisting of a requirements specification and the corresponding solution), and that inductive ML techniques can be used to access them.

With satisfactory representations of the design task and an archive of examples already established, testing the validity of this hypothesis involves the repeated application of the following steps:

1. A model of the design process in question must be proposed. This model depicts the steps necessary for generating a solution from a requirements specification. The content of the heuristic knowledge required to implement this model can then be identified: this is the knowledge to be machine-learned. It should be noted that this model is itself heuristic knowledge; in the context of the work reported here, this knowledge has, once again, been suggested through familiarity with the task.

https://doi.org/10.1017/S0890060401153047 Published online by Cambridge University Press

2. An inductive ML algorithm is chosen for the task of learning each identified element of heuristic knowledge. The relevant training data are extracted from the examples (and, where necessary, redescribed to comply with the format expected by the chosen algorithm).
3. For each learning task, appropriate training parameters are selected and the algorithm is applied to the training data.
4. When all the learning tasks have been completed satisfactorily, a design system can be produced which implements the model of the design process. This system can then be tested and analyzed.

Hopefully, the analysis of a number of systems constructed in this manner will allow some conclusions to be drawn about the viability of this approach to acquiring conceptual design synthesis knowledge in general. It should be noted that a number of "working hypotheses"—such as the design model, the suitability of a particular algorithm, training parameters—are introduced during these five steps. A poor choice for any one of these elements might be enough to condemn the resulting design system to poor performance. In light of this, and although the testing of the individual systems is an essential component of this methodology (and will be discussed below), it should not be concluded from the poor performance of one particular system that the ML algorithm used is unsuited to the task. Rather, in such circumstances, it is necessary to consider the form, content, and behavior of the learned knowledge at a deeper level, and to try to determine whether, under favorable conditions, the approach *could* capture the sort of heuristic knowledge required.

5.1. Design system testing

The customary method of testing the results of machine learning involves partitioning the available examples into distinct training and testing data sets. When applied (automatically) to the testing set, if the learned knowledge suggests the expected output values to all (or a high proportion of) the inputs, then the knowledge is considered satisfactory, and the learning process a success.

This approach presents a number of difficulties in design contexts, though. Rather than simply deciding that a particular design solution is wrong, interest here lies as much in the *degree* to which it fails to meet the specification. A system that generates a high proportion of "nearly right" solutions is more interesting and worthy of further investigation than one that consistently produces utterly wrong solutions. Secondly, there may be two or more equally valid design solutions to a particular design specification. Hence, perfectly good solutions suggested by a system might not correspond to those suggested in the testing set, and, as a consequence, the system appears poor. Finally, with few examples available, setting aside a portion for testing purposes seems to be rather extravagant.

A second approach to testing involves the human appraisal of the suggested solutions. This can be time-consuming, since many test cases must be considered to acquire some accurate reflection of overall performance, and it requires someone with the necessary expertise to be able to judge the quality of the designs. However, given the difficulties of conventional testing, it soon became apparent that such an approach was needed here. Consequently, a manual testing method, which would give some indication of the relative merits of the implemented design systems, was developed.

5.1.1. Testing method

The testing method is based on human appraisal of the system's responses to 20 test cases. Each test case consists of a requirements specification described according to the formal representation: hence, each case is described by 14 parameter values. No test case shares a specification with an example in the archive. Each of the test cases is presented in turn to the system, and the system response recorded. The output is manually evaluated and characterized as falling into one of the following categories:

1. The solution is wholly correct.
2. The solution is over-engineered—it meets the requirements, but contains elements duplicating required functionality.
3. The selected solution elements are correct, but the solution is incomplete. Some requested functionality is not provided by the solution.
4. A correct and complete circuit is a subset of an incorrect solution. The solution provides all the functionality demanded by the specification—but also functionality beyond that requested.
5. Some of the selected solution elements are correct, some are incorrect, and some necessary elements are missing. In other words, some of the requested functionality is provided and some is missing—and some unwanted functionality is also provided.
6. The solution is totally wrong—none of the desired functionality is provided.

In general, these categories are ordered—the lower the number of the category, the better the solution. However, the classification method is subjective, and does not take into account the *extent* to which a solution within a particular category is unsatisfactory. This should be borne in mind and the testing results given in subsequent sections should be viewed with a certain amount of caution—hopefully, however, they should give some impression of the relative merits of the approaches.

The following sections of this paper describe three different systems that have been implemented; these are in-

https://doi.org/10.1017/S0890060401153047 Published online by Cambridge University Press

tended to be illustrative of the type of experimentation that the authors have performed. In the construction of each of these systems, a different type of ML algorithm has been applied; these are an *artificial neural network* algorithm, a *classification knowledge induction* algorithm, and a *conceptual clustering* algorithm. While each of these algorithms operates inductively, they differ in their expression of their learned knowledge.

6. ARTIFICIAL NEURAL NETWORK APPROACH

An artificial neural network (ANN) comprises a network of mathematical units. Each unit in the network is capable of accepting some input signal, processing it according to some internal function, and transmitting the resulting output signal. While the individual units are relatively simple, when connected into a network they can display complex behavior. Each connection between units has an associated numerical weighting which serves to "amplify" or "diminish" the strength of the numerical signals passed along it. The sum of the weighted signals received by a unit determines the strength of the signal that it, in turn, transmits to other nodes.

Different arrangements—or *topologies*—of units and connections exhibit different characteristics. One commonly used network topology is that of the *feed-forward* network. In this some units are designated as "inputs", receiving some initial stimuli from the external environment, while others are designated "outputs" indicating the final response of the network when the sequence of unit calculations has settled into equilibrium. The input units together form the input *layer* of units, and the output units the output layer. Between the inputs and the outputs may be any number of additional layers. Each may contain any number of units, and, in general, a unit in one layer will be connected to all those in the subsequent layer.

A feed-forward network may be "trained" to respond to certain input patterns by producing some associated output. One algorithm for doing this is the *backpropagation* algorithm (Rumelhart et al., 1986). Using this, the network is trained by repeatedly presenting examples of the correct combinations of inputs and outputs to the network, and gradually altering the connection weightings so that, for every example, the input produces the desired network output. It is found that, under favorable conditions, an ANN trained in this manner has learned a generalized representation of these associations. In other words, the ANN can respond with appropriate outputs to input patterns other than those used during training.

ANNs are *subsymbolic* learners: they expect data to be represented numerically [and typically as normalized values between 0 (or -1) and 1]. Learning parameters govern, amongst other things, the rate at which weightings are modified, and the point at which training is considered complete. To learn more complex associations, ANNs tend to need greater numbers of intermediate units—and more layers of these units.

6.1. Artificial neural networks and design synthesis knowledge

The heuristic design knowledge is considered to exist in the form of associations between design requirements and solution elements. Since ANNs are able to learn the generalized associations that hold between sets of inputs and sets of outputs, they would appear to be well-suited to acquiring this sort of knowledge, and, consequently, they have been investigated in the course of the research reported here. In addition, ANNs have been seen to cope adequately with a certain degree of noise in their data, which, given the state of the design archive, may prove a valuable characteristic in this case.

To attempt to exploit these appealing properties, the authors have applied ANNs within the framework of a number of different models of the fluid power circuit conceptual design task. These models vary greatly in complexity and detail; the next section describes the simplest of these models.

6.2. Model of the design process

The model comprises a single stage: A given requirements specification is transformed in one pass into an appropriate description of the required solution elements. This transformation would be achieved using a trained ANN having inputs corresponding to the requirements parameters and outputs corresponding to the solution elements. When coupled with the circuit template, the selected solution elements form a complete description of the solution. This model is shown in Figure 3.

To implement this model, an ANN would be trained upon the archive of designs to recognize the associations between the requirements and solution elements. The trained ANN would, hopefully, generalize enough to be able to respond to a wide range of design problems, and not merely those in the archive.

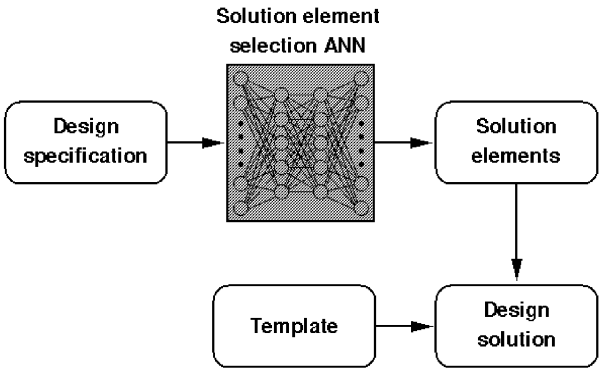


Fig. 3. An ANN-based model of the design process.

https://doi.org/10.1017/S0890060401153047 Published online by Cambridge University Press

6.3. Data preparation and algorithm application

ANNs can deal only with numerical data: Hence, before training can commence, the archive examples must be re-expressed in such terms. This raises the issue of how such a transformation is to be effected without loss of information, or the introduction of undesirable biases into the system. To do this, consideration must be made of the different types of parameters in the representations. In this case, the adopted representation of the design requirements contains two types of parameters, namely, binary-valued "functional" parameters and ternary-valued "context" parameters. The binary-valued parameters were encoded in terms of a single input flag, having a value of 1 for "yes" and a value of 0 for "no".

The ternary-valued context parameters (such as the maximum magnitude of load to be moved), however, suggest a greater number of alternative representations. Once again, a single node might be used, with, say, 0 representing "low", 1 representing "high," and 0.5 representing "medium". Or else, three separate nodes could be used, each corresponding to one of the possible values (although using three mutually exclusive units to represent a single concept can be difficult to justify, both semantically and in terms of the extra bias that this may introduce into the network behavior towards this parameter). Here, the decision was taken to encode using a single flag having a value of 1, 0.5, or 0.

Similar difficulties could potentially surround the encoding of the solution elements. Here, it is necessary to indicate the presence or absence of each of the distinct solution elements. In terms of the ANN outputs, this lends itself to the use of a separate unit corresponding to each element, with the value 1 indicating its presence and 0 its absence. This method was adopted here. Given these encodings for requirements and solution elements, the archive examples could now be redescribed accordingly to form the ANN training data [see Bishop (1995) for a more detailed discussion of the possibilities for and implications of different encoding schemes].

These encodings also implicitly define some of the ANN topology; since 14 encoded parameters describe the requirements, and 17 encode the solution, the ANN would need 14 inputs and 17 outputs. The next step is to define the remainder of the topology (that is, the numbers of additional network layers and the numbers of units in each) and values for the training parameters. There is little assistance for the selection of these. Most attempts to use ANNs involve a "trial and error" approach—the correct topology and parameters have been selected when the trained network performs satisfactorily. This was done here using the "manual" testing method described above; from a number of trained ANNs, the one that displayed the "best" performance was selected.

However, before the performance could be judged, it is necessary to interpret the output of the ANNs in some manner. When trained, the outputs of the ANNs are not constrained to take values of 0 or 1: each can assume any value between the two. When this happens, some interpretation must be applied to the each output. A value of, say, 0.64, may be taken to be some indication of the likelihood of the corresponding element being present. However, at some stage designs have to consist of definite elements, so a decision must be made about whether the element should be included or not. A second approach is to apply a hard threshold, asserting, for example, that elements assigned values of above 0.5 are definitely included in the solution, with all others excluded (this is the approach adopted here). Another option would be to use some combination of thresholds and likelihoods to generate a number of alternative solutions.

6.4. Results

The results of testing the system implemented using the "best" trained ANN on the 20 test cases are shown in the graph below (Figure 4). In this case, four solutions were considered to represent category 1 solutions (the "best" category). Focusing instead on the "worst" solution categories, 5 and 6, it can be seen that, respectively, 5 and 1 suggested solutions fall into these categories.

6.5. Discussion

In general, the ANNs were found to be able to learn certain, simple associations between specification attributes and solution elements. Typically, these learned associations were those which seemed to be well-exemplified in the data. However, network performance was, on the whole, poor. The most obvious reason for this poor performance is a lack of data: 30 example cases would generally be considered a number of orders of magnitude too few for training an ANN to learn about a problem of this complexity. In an attempt to address this perceived problem, systems were constructed using a number of more complex models of the design process. These models, in various ways, decompose the process into simpler subtasks; it was hoped that fewer data would then be required to train an ANN to perform each

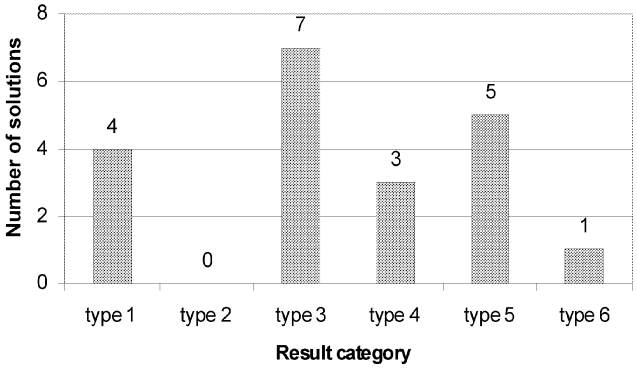


Fig. 4. ANN-based system test results.

https://doi.org/10.1017/S0890060401153047 Published online by Cambridge University Press

subtask. These systems were found to give slightly improved performance (at the cost of acquiring and introducing more knowledge in the process models), but, even so, the number of available training data seems too few to learn satisfactorily the heuristic knowledge.

However, since it seems very unlikely that the examples in the archive cover the full range of associations, any attempt to learn complete, generalized synthesis knowledge for the task, regardless of approach or algorithm, seems bound to fail. With ANNs, though, it is difficult to determine just what has and has not been learned, since the numerical units and weighted connections of the trained ANN are not particularly susceptible to analysis (although some work—such as that described by Fu (1994)—has been devoted to the task of extracting rules from trained ANNs). Another problem particular to ANNs arises since they are fully defined (albeit with random connection weightings) from the outset of training, and so, even at this stage, they will produce (almost certainly incorrect) responses to an input pattern. The aim of the training algorithm is to alter these weightings so as to represent the associations exemplified in the training data, but nothing in the *appearance* of the networks themselves changes during training to reflect this move from being an untrained to being a trained network. Hence, even when undertrained, an ANN will respond to every input pattern with some output—here, a design solution will be produced in response to every set of requirements. In cases where this response is based on insufficiently modified weightings, it might be preferable if the network were to fail to produce a solution, or to otherwise indicate a lack of confidence in its response.

6.6. Issues

Based on this experimentation and analysis, a number of general issues surrounding the attempt to use ANNs to acquire design synthesis knowledge have been identified. These are as follows:

- It will often be necessary to encode a problem into the form expected by a specific ML algorithm. However, this task seems particularly difficult in this case, since ANNs can manipulate only numeric information (and this within, usually, a restricted range), while design problems typically contain at least some qualitative information. This makes it difficult (and probably impossible) to achieve the encoding of design problems without introducing some unintentional biases. In addition, some interpretation must be imposed upon the network outputs to make sense of their real-valued outputs.
- The problems of testing design knowledge mean that it is particularly difficult to alight on suitable ANN topologies and learning parameters.
- When trained with insufficient data, a network may conceivably contain "bad" knowledge, initial knowledge that has not been modified correctly. This can result in poor designs. However, there is no indication when such knowledge is being applied: This is a perilous state of affairs in design contexts.
- The trained network consists of a set of units and their weighted connections. It is difficult to learn much about the acquired design knowledge through analysis of these.

In an attempt to discover more about the extent and content of the knowledge that could be learned from the archive, attention switched in the investigation to consideration of ML algorithms that express their knowledge in forms that are immediately comprehensible to humans.

7. CLASSIFICATION KNOWLEDGE INDUCTION APPROACH

A large number of the available inductive ML algorithms that express their knowledge in comprehensible forms are tailored towards the acquisition of *classification* knowledge, that is, knowledge which will allow some event to be appropriately classified into one of a number of categories. Hence, a different approach was to try to extract synthesis heuristics in the form of classification knowledge from the design examples.

The algorithm used in this case is CN2 (Clark & Niblett, 1989; Clark & Boswell, 1991), which expresses its learned knowledge in the form of classification rules. In other words, given a number of examples, each of which consists of a set of attribute values and the appropriate classification of this set, this algorithm proceeds in an iterative fashion to find a good set of *if–then* rules for classifying the data. This is performed by starting with a set of rules having conditions (a rule condition takes the form of a test for particular attribute values) of a very general nature, testing these, and gradually specializing the better rules in an attempt to find still better ones.

A rule is tested according to two measures. First, an error measure is applied; this measure favors rules that "explain" a large number of examples of a single class and few of other classes. Second, the *significance* of the rule is calculated—a rule is considered significant if it describes a regularity in the data that is unlikely to have occurred by chance. Less significant rules are only weakly suggested by the evidence, and so should be adopted into the set of rules with greater caution.

Once they have been derived, the rules may then be embedded within an expert system shell. A shell provides the inference and control mechanisms that permit the application of the rules to some new data, in this case, to a new requirements specification.

7.1. CN2 and conceptual design synthesis knowledge

So, since CN2 is able, like ANNs, to produce generalized knowledge from individual examples, and, in addition, the

https://doi.org/10.1017/S0890060401153047 Published online by Cambridge University Press

knowledge that it acquires is more comprehensible than ANNs, it was decided to investigate its capacity for learning design synthesis knowledge. A further advantage, in this particular case, is the fact that CN2 manipulates symbolic data (that is, data represented by labels such as "yes" or "high"), the form in which the archive information is already expressed.

The principle drawback with using CN2 and similar algorithms in this context is the nature of the knowledge it learns. Complex design tasks cannot be thought about easily in terms of classification rules. However, if the use of CN2 was to be investigated, these are the terms in which the task needs to be recast.

7.2. Model of the design process

The devised model is as follows: Given a requirements specification, the design process can be thought of as one of determining whether each solution element, in turn, falls into either the class "present" or else the class "absent". Each of these decisions would be made by a set of classification rules relevant to that solution element. The condition part of each of these rules would take the form of a test of the values of some requirements parameters, with the conclusion indicating, in the event of the condition being met, whether or not the element should be present in the solution. This model of the process is shown in Figure 5.

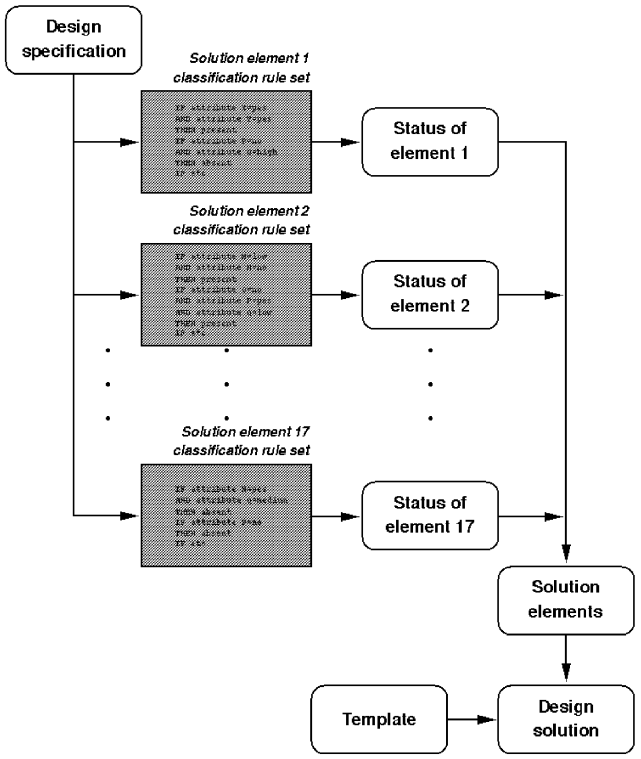


Fig. 5. A classification knowledge-based model of the design process.

7.3. Data preparation and algorithm application

To implement this model, 17 sets of classification rules—one for each solution element—would be needed. To learn each, CN2 would require a different training data set extracted from the archive. Each training set contains 30 patterns, one corresponding to each of the 30 archive design examples. Each of these patterns contains the requirements of the example (described simply in terms of their values) along with a label indicating if the solution element in question is "present" or "absent" in the corresponding design solution.

The selection of appropriate learning parameters is easier for CN2 than it is for ANNs. A parameter governs the size of CN2's "working memory" which stores potential rules. A larger memory results in a greater time required for learning, but potentially more effective rules. Here, as training times are of no consequence, the maximum setting was allocated to the parameter.

The algorithm was applied to the training data and, as a consequence, 17 rule sets, one per solution element, were created. As an example, the rule set for the solution element *POCV_A* is as follows:

```
IF   hold_on_failure = no
THEN element POCV_A = absent [18]
IF   smooth_accelerations = yes
THEN element POCV_A = absent [7]
IF   motor_required = yes
AND  energy_efficiency_paramount = no
THEN element POCV_A = absent [7]
IF   smooth_accelerations = no
AND  hold_on_failure = yes
AND  motor_required = no
THEN element POCV_A = present [6]
IF   smooth_accelerations = no
AND  max_load = high
THEN element POCV_A = present [1]
```

This element (a pilot-operated check valve) is generally used to hold a load in position, especially in response to system failure. Attached to each rule is an indication of the number of archive examples that the rule "explains"—the greater this number, the greater should be the confidence in the rule being a good one. This information is later used during the design process: If more than one rule for a certain solution element is found to be applicable given the new design specification, then the rule chosen is that in which there is the greatest confidence.

The classification rules were placed within an expert system shell along with appropriate control structures to form an automated design system [the *CLIPS* shell (Giarratano & Riley, 1997) was that used in this instance]. When a new set of requirements is supplied to the system, the shell finds the "best" applicable rule for determining the status of each solution element. This is the rule having a condition that accords with the parameter values of the new requirements,

https://doi.org/10.1017/S0890060401153047 Published online by Cambridge University Press

and which explained the greatest number of the training examples. By applying the best rule for each element, the solution is constructed, consisting of those elements classified as being present. If no applicable rule is found for a particular solution element, then the status of the element remains undetermined at the completion of the process (and hence, the design process remains unfinished).

7.4. Results

The performance of this system, in general, seems better than those produced using ANNs. The results across the entire test set (Figure 6) indicate that over half of the solutions fall into category 1. Only one solution falls into either of the worst two categories.

7.5. Discussion

The design strategy applied in this model would not seem to be a natural reflection of the process, and as such remains unsatisfactory. Design solution synthesis would seem to require a more "holistic" approach than the piecemeal consideration of solution elements. That said, simplifying the task by describing it in terms of a series of independent classification tasks may represent a practical approach to learning about complex tasks for which few data are available.

Again, the inadequacies of the data are evident in the performance. Some of the rule sets are obviously not general enough to cope with all situations, a trait suspected of the knowledge learned by the ANNs, but concealed from view. Some combinations of requirements values do not match the conditions of any of the rules in a set; when this happens, no rule is applied, and the status of the corresponding solution element at the completion of the process remains undetermined.

There is an additional advantage with this approach in that the learned heuristics can be inspected and understood. This means that the learned knowledge can be assessed independently of any assessment of the system in which it is embedded (by, for instance, showing the rules to domain experts). Certainly, many of the learned rules do seem to be reasonable in the domain. However, CN2 does learn some "poor" rules, which, when applied, produce inappropriate solutions. The measure of confidence the algorithm attaches to each of the rules it generates is of some use here, since these poor rules tend to be supported by few of the examples. However, with 17 solution elements and only 30 examples, there are often very few examples of, in particular, the presence of a particular element in a solution. Consequently, the rules indicating that an element is "present" will rarely be supported by a great deal of evidence.

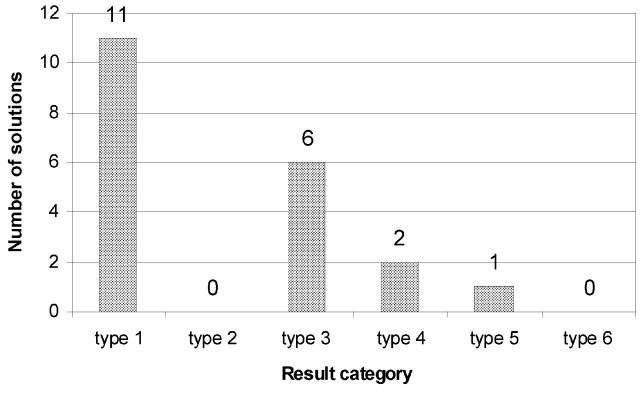


Fig. 6. Classification rule-based system test results.

7.6. Issues

CN2 is just one of many different symbolic learning algorithms which learn classification knowledge inductively. These algorithms operate by recognizing patterns in their training data in different ways, and represent their learned knowledge in different (but usually comprehensible) forms—as, for instance, rules, as in the case of CN2, or as a decision tree. These differences in their operation result in different characteristics in performance, with some algorithms being more robust in the face of missing data values, for example. However, their underlying similarities mean that a number of issues raised by the use of CN2 described here are applicable in general to this whole family of algorithms when trying to learn design synthesis knowledge:

- The use of a classification knowledge-learning algorithm involves redescribing the design problem in terms of one or more classification tasks. This does not seem a very natural or appealing way of describing conceptual design problems. This approach results, in this case, in decisions about each solution element being made without reference to the decisions being made elsewhere; this would seem to be too reductionist an approach to what is a complex task.
- Again, with what would seem to be too few examples to provide sufficient coverage of the range of design problems, the algorithm will be unable to infer complete synthesis knowledge. This becomes evident in this case when the status of an element remains undetermined after the consideration of some new set of requirements. However, this inadvertent declaration of ignorance is preferable to the response of ANNs in similar situations. Furthermore, since the knowledge is comprehensible, it can be examined, corrected, and augmented, and as such might form a useful starting point in a wider knowledge-acquisition process.

8. CONCEPTUAL CLUSTERING APPROACH

Conceptual clustering algorithms attempt to produce some sort of description of a domain from a set of data exempli-

https://doi.org/10.1017/S0890060401153047 Published online by Cambridge University Press

fying it; typically, this description takes the form of a taxonomy or hierarchy. Items towards the top of this hierarchy are more general domain concepts—the single node at the top of this hierarchy (the *root* node) will be the most general concept, describing all of the data. Nodes below the root get successively more specific, referring to more specialized concepts within the data. Nodes at the same level in the hierarchy describe alternative concepts at that level. Terminal nodes describe the most specific concepts—usually, these will be the examples themselves. These algorithms work by searching for similarities within the data. Examples that share some similarity are clustered together, and a description of this similarity is used to describe the concept so formed.

One example of a clustering algorithm is COBWEB (Fisher, 1987). COBWEB attempts to construct a conceptual hierarchy from a set of examples. Each concept, or "category", in the hierarchy is described in terms of the probabilities of examples of that concept possessing particular attribute values. In contrast to the algorithms described previously, COBWEB is an *unsupervised* learning algorithm: The hierarchy construction is based on the similarities amongst the attribute values of the data—no prior classification of the examples is required.

The algorithm functions in an incremental manner. The first example is used to define a root node. Each subsequent example is added to the root, modifying the description accordingly. It is then recursively classified into one of the child categories, or, if the example is not sufficiently similar to an existing category, by creating a new one to house this example. Eventually, a category describing this example alone will form a terminal node in the hierarchy. Additional operators can merge two categories into a single category, or split a category into several. These help to revise the hierarchy in the light of new examples that have the effect of invalidating the classifications of previous examples.

8.1. COBWEB and conceptual design synthesis heuristics

The use of a clustering algorithm to try to capture synthesis heuristics has been suggested by Reich and Fenves (1992), amongst others. In effect, the algorithm is used to learn the similarity measure of a Case-Based Reasoning-type (CBR) design system. To investigate the merits of this approach, the authors of the present paper have selected the COBWEB algorithm as being an algorithm typical of this type. Moreover, this algorithm works incrementally: New examples can be added to the existing hierarchy when they become available. As such, the algorithm can incorporate new examples as and when they arrive (unlike most ANNs and classification learning algorithms), and so, might be thought to be better suited to the continuing and evolving nature of design.

8.2. Model of the design process

The CBR approach to design has been enthusiastically adopted by a number of researchers (Maher & Gomez de Silva Garza, 1997). In short, this approach relies on the existence of some "memory" of previous design episodes—such as the design archive. When a new design problem is encountered, the design strategy is one of retrieving that design solution from memory that meets a requirements specification that is "most similar" to the current requirements—and this solution is proposed as the solution to the new problem. However, if, as here, the design memory is less than comprehensive, it is unlikely that this solution will be adequate in its retrieved form—in which case, it will need to undergo some modifications in order to be acceptable (this modification phase is not investigated in this experiment).

The heuristic knowledge necessary to implement this process consists of the knowledge of how to determine which archive example has a requirements specification that is most similar to the new set. The idea here is that this knowledge can be expressed in the form of a conceptual hierarchy of the sort learned by COBWEB. A hierarchy is formed of the archive examples, based upon the description of their requirements values alone. A new requirements specification can be classified through this hierarchy, eventually being classified into one of the terminal nodes. This node will correspond to the requirements of a particular archive example. Since this requirements specification has been judged in this way to be similar to the new problem, the corresponding solution from the archive example is proposed as (the basis of) the solution to the new problem. This model of the system is depicted in Figure 7.

8.3. Data preparation and algorithm application

COBWEB is an unsupervised algorithm, searching for similarities implicit within the descriptions of examples, rather than relying on explicit output or class information. Hence, the training data consists of 30 archive examples: Each is described in terms of its requirements values alone. Since the algorithm operates on qualitative, symbolic labels, once again, the requirements are expressed simply using their archive values.

8.4. Results

A hierarchy was generated using COBWEB, and a design system implemented using it. However, at this stage, the experiment was halted, without any methodical testing of the system, for reasons that will now be discussed.

8.5. Discussion

The generated hierarchy could be used as described above to classify a new set of requirements and indicate a design solution. This approach has the advantage of returning a

https://doi.org/10.1017/S0890060401153047 Published online by Cambridge University Press

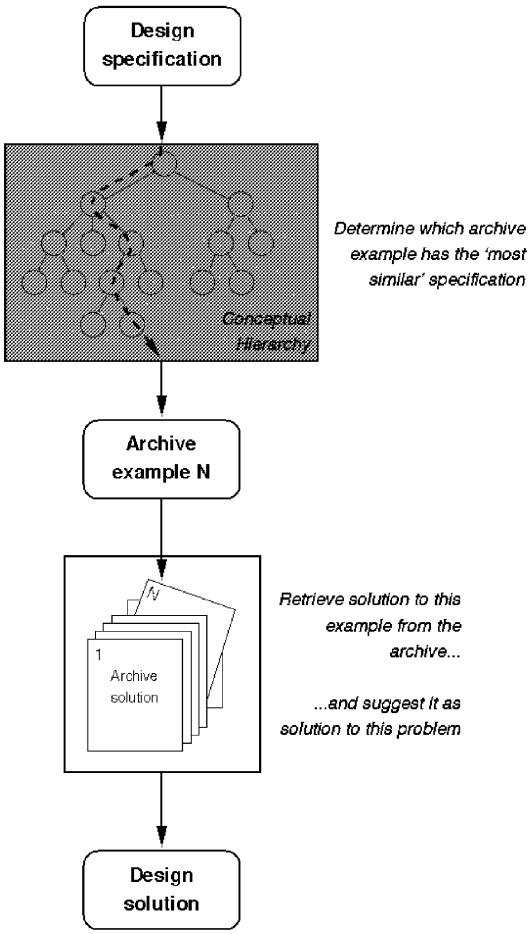


Fig. 7. Conceptual hierarchy-based model of a design system.

complete, fully configured fluid power circuit (bearing in mind that the limited number of examples means that it is unlikely that the returned solution will be wholly adequate).

While this might seem an attractive approach, a major theoretical objection can be raised against such a method. Since it forms the classification hierarchy based solely upon the requirements of the available design examples, this hierarchy merely reflects the similarity patterns amongst the (limited number of) problems found in the archive. It cannot be said to be a hierarchy *for design*, since it incorporates no information about the subsequent use of the values of the requirements parameters during the design process. There is no reason to believe that the criteria for judging similarity embodied in this hierarchy is the appropriate heuristic knowledge for a CBR design system. The ML approaches described earlier had used some concept of the "purpose" of the requirements inasmuch as they employed supervised learning, and so the data referred to not only requirements parameters, but also to the corresponding solution elements. The most that can be said about this hierarchy is that it reflects the similarities and biases amongst the requirements of the available archive data. Accordingly, the use of conceptual clustering techniques such as COBWEB was investigated no further.

8.6. Issues

The analysis of the shortcomings of this approach allow several general issues to be raised:

- Whilst the case-based reasoning approach to design is appealing, and this algorithm might seem to provide a method of selecting a similar design from the available examples, the hierarchy is produced based purely on the similarities of requirements values. There seems no reason to suggest that a hierarchy generated by COBWEB or any similar clustering algorithm is useful for the design task.
- The usefulness of any unsupervised learning algorithm to acquire conceptual design synthesis knowledge, regardless of the particular model of the process adopted, would seem to be limited. At its core, design is a purposeful activity; the purpose of the designer is to generate the description of an artefact that meets the requirements. Unsupervised algorithms, by their very nature, search for internal associations regardless of any ulterior purpose: The notion of design synthesis as a process is not communicated to them, and so is not reflected in their learned knowledge.

9. MACHINE LEARNING DESIGN HEURISTICS—ISSUES AND DISCUSSION

The preceding sections ought to have conveyed something of the nature of the authors' investigations. In the course of this work, a number of general issues have arisen regarding this approach to learning conceptual design synthesis heuristics. These will now be described and discussed.

9.1. Representation of the task

To describe and reason about design tasks on computers, it is necessary to represent the task in a formal, consistent manner. The production of these representations is not a trivial matter: Poor choices made at this stage may hinder the performance of the system or, if applying ML, obstruct learning about the task.

9.2. The model of process

To provide a context for the use of ML algorithms, it is necessary to hypothesize a model of a design process for the task in hand. This model takes the form of a design strategy, for which the required heuristic knowledge should be apparent. The model will be subjective to a certain degree, and, perhaps, influenced unduly by the desire to use a particular ML algorithm. The only way to wholly vindicate

https://doi.org/10.1017/S0890060401153047 Published online by Cambridge University Press

a particular model is to produce a complete, correct automatic design system based upon it.

9.3. Encoding

The training examples must be redescribed in terms appropriate to each particular ML algorithm, while preserving as far as possible the information contained within them. This is not always a trivial task: A bad choice of encoding can make the learning task more difficult or even impossible; however, picking a good encoding often requires a certain degree of knowledge about the problem in hand.

9.4. Testing

For systems that are not complete and correct, it is difficult to attach some measure of relative "goodness". However, this must be done, if progress is to be made towards this ideal system. The response from some knowledge sources (such as in the case of the trained ANN in the above experiments) must be interpreted before it can be examined, which may not be a simple task.

9.5. The opacity of the learned knowledge

The knowledge learned by, in particular, ANNs is virtually incomprehensible. In design contexts, and given the difficulties of testing, it would be useful to examine the sorts of relationships that are being learned and, just as importantly, those that are not. While the form of the knowledge learned by symbolic algorithms such as CN2 might seem to make them less appropriate, the intelligibility of this learned knowledge can make them more useful for investigating the learning that is taking place, the adequacy of data, and the nature of the learning task itself. This characteristic also allows some rationale to be provided for the decisions taken, which may be necessary if automated design systems are to gain widespread acceptance.

9.6. The aptness of the learned knowledge

It is important to ensure that the type of knowledge generated by an algorithm is that which is required to implement the adopted model of the design process. The application of COBWEB described above provides an example of an algorithm producing inappropriate knowledge: While it generates a hierarchy expressing some notion of similarity, this is not the similarity metric required to implement a CBR system. Unsupervised learning algorithms would seem, in general, to be of limited use for learning design synthesis knowledge, since they discount the relationship between requirements and solutions which is fundamental to such knowledge.

9.7. Training data

The inductive ML algorithms that are currently available are relatively unsophisticated. Typically, these algorithms have been developed and tested using learning tasks having a complexity much lower than that of learning about design. Put in simple terms, all supervised learning algorithms attempt to learn the function that governs the mapping of inputs onto outputs. The training data is expected to consist of a number of examples of the application of this function, from which, it is hoped, it can be inferred. Hence, for learning to be successful, it is necessary that the relevant inputs and outputs are available, and that the examples represent mappings consistent with this function, and furthermore, that the examples should be available in sufficient quantity to allow generalization across the range of possible inputs. The effectiveness of these algorithms when set learning tasks that do not share these characteristics is doubtful.

Design problems would appear to share few of these features. The design examples that have been collected are from a variety of sources and have been generated by different individuals. Although they may be examples of the same task, in all likelihood they are products of different design "functions", and display different understandings of the relationships between requirements specification and solution: Different designers can produce different solutions, equally valid, to the same design problem. Furthermore, design domains are not consistent: New products are introduced and new practices encouraged. The design task cannot be circumscribed simply, if at all, with all the relevant inputs and outputs identified. There are relatively few examples available, and those are often poorly documented. Rather than being some objective, mechanistic procedure, design is a process performed by humans for humans, with all the nuances and subtleties that this entails.

While this might paint a bleak picture for the future of ML algorithms in such applications, it should be remembered that, under the same conditions, human designers manage to learn how to design. The emphasis in ML research to date has been placed on the development of general-purpose, domain-independent algorithms. What these difficulties seem to demand are design-specific (and perhaps also domain-specific) learning algorithms, able to cope with the particular character of available design examples, and better suited to representing design problems. Rather than attempting to learn in a vacuum, unaware of the domain beyond the description of the examples, such algorithms would have recourse to other, analytical knowledge of the domain to guide their learning, as well as access to wider "common-sense" knowledge.

With the techniques that are currently available, the most effective use of inductive ML in the acquisition of design synthesis knowledge might be as but a single component in a wider knowledge-acquisition exercise. As seen above with the application of CN2, the use of a symbolic classification

https://doi.org/10.1017/S0890060401153047 Published online by Cambridge University Press

technique produces a comprehensible knowledge base. Whilst this knowledge base may not be complete, nor correct, it may serve as a useful starting point for further knowledge acquisition with human design experts; they could be asked to analyze and comment on the knowledge, and suggest corrections and additions, thereby gradually revising the knowledge base towards one that can be used with a greater degree of confidence.

10. CONCLUSIONS

The quality of automatic design synthesis tools is dependent on the quality of the knowledge they contain. Conventional knowledge-engineering approaches to capturing heuristic knowledge for such computer systems have been found to be lacking. Inductive ML has been suggested as an alternative by which this knowledge might be acquired. To date, however, there has been little in the way of documented research into the feasibility of such an approach in general for capturing design heuristics—and that which has been reported in the literature is inconclusive, being based, in general, on artificial design problems and data.

To attempt to rectify this situation, the authors have conducted a number of experiments into the use of inductive ML to capture the heuristic knowledge necessary to synthesize conceptual designs of fluid power circuits. Algorithms of three distinct classes, representative of available inductive ML techniques, have been applied to this learning task. This experimentation would seem to suggest that, without recourse to additional techniques, this is not a viable approach for acquiring the heuristics for real-world design problems. The disparity between the complexity of human learning processes and the unsophisticated state of current ML techniques would seem to be too great.

However, when one remembers the relative immaturity of machine learning research, it is, perhaps, only to be expected that these algorithms should be found wanting when faced with learning about something as complicated and demanding as conceptual design. Identifying these deficiencies would seem to be a necessary step in the development of more sophisticated approaches that are able to cope with the complexities of engineering design.

ACKNOWLEDGMENTS

The research reported here was undertaken in the Engineering Design Centre at the University of Bath, U.K., funded through UK EPSRC grant GR/L26858. The authors would like to thank the anonymous reviewers for their perceptive comments and suggestions for improvement of the original manuscript.

REFERENCES

Berry, D.C. (1987). The problem of implicit knowledge. *Expert Systems 4(3)*, 144–151.

Bishop, C.M. (1995). *Neural Networks for Pattern Recognition*. Clarendon Press, U.K.

Clark, P., & Boswell, R. (1991). Rule induction with CN2: Some recent improvements. In *Machine Learning—Proceedings of the Fifth European Conference (EWSL-91)*, (Kodratoff, Y., Ed.), pp. 151–163. Springer-Verlag, Berlin.

Clark, P., & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning 3(4)*, 261–283.

Coyne, R.D., Newton, S., & Sudweeks, F. (1993). A connectionist view of creative design reasoning. In *Modeling Creativity and Knowledge-Based Creative Design*, (Gero, J.S. and Maher, M.L., Eds.), pp. 177–209. Lawrence Erlbaum, New Jersey.

Duffy, A.H.B. (1997). The 'what' and 'how' of learning in design. *IEEE Expert Magazine* May/June 1997, 71–76.

Duffy, A.H.B., Brown, D.C., & Maher, M.L. (1996). Special issue: Machine learning in design. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 10(2)*, 81–82.

Duffy, A.H.B., Brown, D.C., & Goel, A.K. (1998). Special issue: Machine learning in design. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 12(2)*, 93–84.

Duffy, S.M., & Duffy, A.H.B. (1996). Sharing the learning activity using intelligent CAD. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 10(2)*, 83–100.

Fisher, D.H. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning 2(7)*, 139–172.

French, M.J. (1985). *Conceptual Design for Engineers*. 2nd Edition, Springer-Verlag, London.

Fu, L. (1994). *Neural Networks in Computer Intelligence*. McGraw-Hill, New York.

Giarratano, J.C., & Riley, G. (1997). *Expert Systems: Principles and Programming*. 3rd Edition, PWS Publishing, Boston, Massachusetts.

Gillies, D. (1996). *Artificial Intelligence and Scientific Method*. Oxford University Press, Oxford.

Hart, A. (1988). Knowledge acquisition for expert systems. In *Knowledge, Skill and Artificial Intelligence*, (Göranzon, B. and Josefson, I., Eds.), pp. 103–111. Springer-Verlag, London.

Ivezic, N., & Garrett, J.H., Jr. (1994). A neural network-based machine learning approach for supporting synthesis. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 8(2)*, 143–161.

Lenat, D.B. (1983). The role of heuristics in learning by discovery: Three case studies. In *Machine Learning: An Artificial Intelligence Approach*, (Michalski, R.S., Carbonell, J.G., and Mitchell, T., Eds.), Vol. 1, pp. 243–306. Morgan-Kaufmann, Los Altos, California.

Lu, S.C.-Y., & Chen, K. (1987). A machine learning approach to the automatic synthesis of mechanistic knowledge for engineering decision making. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 1(2)*, 109–118.

Maher, M.L., Brown, D.C., & Duffy, A.H.B. (1994). Special issue: Machine learning in design. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 8(2)*, 81–82.

Maher, M.L., & Gomez de Silva Garza, A. (1997). Case-based reasoning in design. *IEEE Expert Magazine*, March–April 1997, 34–41.

Maher, M.L., & Li, H. (1994). Learning design concepts using machine learning techniques. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 8(2)*, 95–111.

McDermott, J. (1982). R1: A rule-based configurer of computer systems. *Artificial Intelligence 19(1)*, 39–88.

Michie, D. (1982). The state of the art in machine learning. In *Introductory Readings in Expert Systems*, (Michie, D., Ed.), pp. 208-229. Gordon and Breach, New York.

Mittal, S., & Freyman F. (1989). Towards a generic model of configuration tasks. *Proc. Eleventh Int. Joint Conf. Artificial Intelligence*, pp. 1395–1401. Morgan Kaufmann, Los Altos, California.

Pahl, G., & Beitz, W. (1996). *Engineering Design: A Systematic Approach*. 2nd Edition, Springer-Verlag, London.

Reich, Y., & Fenves, S.J. (1989). The potential of machine learning techniques for expert systems. *Artificial Intelligence for Engineering Design, Analysis and Manufacture 3(3)*, 175–193.

Reich, Y., & Fenves, S.J. (1992). Inductive learning of synthesis knowledge. *International Journal of Expert Systems: Research and Applications 5(4)*, 275–297.

Reich, Y., & Fenves, S.J. (1995). A system that learns to design cable-stayed bridges. *Journal of Structural Engineering 121(7)*, 1090–1100.

Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1986). Learning internal representations by error propagation. In *Parallel Distributed Process-*

https://doi.org/10.1017/S0890060401153047 Published online by Cambridge University Press

ing: Explorations in the Microstructures of Cognition, (Rumelhart, D.E. and McClelland, J.L., Eds.), Vol. 1, pp. 318–362. MIT Press, Cambridge, Massachusetts.
Ullman, D.G. (1997). *The Mechanical Design Process*. 2nd Edition, McGraw-Hill, New York.
Winston, P.H. (1993). *Artificial Intelligence*. 3rd Edition, Addison-Wesley, Massachusetts.

Stephen Potter was formerly a researcher in the Engineering Design Centre within the Department of Mechanical Engineering at the University of Bath. His doctoral thesis is entitled "Artificial Intelligence and Conceptual Design Synthesis".

Mansur J. Darlington is a cognitive scientist working in the Engineering Design Centre at the University of Bath. He has a particular interest in the capture of the design requirement and how the conceptual and language content of design requirement expression can be harnessed to control and formalize the elicitation process for automation.

Steve J. Culley is Head of Design in the Department of Mechanical Engineering at the University of Bath. He has researched in the engineering design field for many years. In particular this work has centered on the provision of information and support to engineering designers. He pioneered research into the introduction and use of the electronic catalogue for standard engineering components and has extended this work to deal with systems and assemblies. He has over 100 publications and is currently in the process of writing a book.

Pravir K. Chawdhry is a lecturer in Engineering at the Department of Mechanical Engineering at the University of Bath. His particular interests lie in the use of computer methods in engineering design, especially for concurrent engineering.

https://doi.org/10.1017/S0890060401153047 Published online by Cambridge University Press